EXHIBIT 2

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

ARISTA NETWORKS, INC.
Petitioner

v.

CISCO TECHNOLOGY, INC.
Patent Owner

Case No.: IPR2016-00119
Patent 7,047,526

DECLARATION OF DOUGLAS W. CLARK

Mail Stop PATENT BOARD
Patent Trial and Appeal Board
U.S. Patent & Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450

Ex. 1014
IPR of U.S. Pat. No. 7,047,526

TABLE OF CONTENTS

I, Douglas W. Clark, hereby declare as follows:

I. Qualifications

1. I received a Ph.D. in computer science from Carnegie-Mellon University in 1976 and a Bachelor of Science in engineering and applied science from Yale University in 1972. Since receiving my doctorate, I have devoted my professional career to the research, design, development, study, and teaching of numerous aspects of computer systems architecture and design. I have studied, taught, practiced, and researched in the field of computer science for over forty years.

2. I am currently a Professor of Computer Science at Princeton University. I have held this position since 1993. I have taught numerous courses at Princeton, including an introductory computer science course, a foundational course for first-year students, and advanced courses in computer architecture for upper-level undergraduates and graduate students. I have also taught a number of advanced graduate seminars about various computer science topics. I have taught most of these courses several times. In addition, in 1990-91, I taught as a Visiting Lecturer in the Division of Applied Sciences at Harvard, and in 2003 I was a Visiting Professor of Computing and Information Science at the University of Pennsylvania.

3. In addition to my experience in academia, I have over 17 years of industrial experience designing computer systems. From 1976 to 1980, I was a Member of the Research Staff at the Xerox Palo Alto Research Center, where I

worked chiefly on the design of the Dorado, one of the earliest high-performance workstations. From 1980 to 1993, I worked for the Digital Equipment Corp., first as a Principal Engineer in the Systems Architecture Group, and then as a Consulting and Senior Consulting Engineer in both the Advanced VAX Systems Engineering and Alpha Advanced Development groups. I worked mainly on the architecture, organization, design, simulation, and performance analysis of VAX and Alpha computers. I was one of the principal designers of the VAX 8700 and VAX 8800 – both highly successful machines of the late 1980's.

4. I have authored or co-authored about 60 refereed academic publications in the fields of computer science and engineering. In addition, I have been a referee or associate editor for the following academic journals: ACM Transactions on Computers, IEEE Transactions on Computers, and IEEE Computer. My curriculum vitae includes a list of publications on which I am a named author.

5. I have also been a program committee member or co-chair at a number of national and international conferences/symposiums, including the International Conference on Computer Design, SIGMETRICS Conference on Measurement and Modeling of Computer Systems, and International Symposium on Computer Architecture.

6. My curriculum vitae, attached herein, contains further details regarding my experience, education, publications, and other qualifications to render an expert opinion in connection with this proceeding.

7. I am being compensated at my normal consulting rate for my work. I receive no other forms of compensation related to this case nor do I have an ownership in either Cisco Systems, Inc. or Arista Networks, Inc. My compensation is not dependent on or otherwise contingent upon the results of this proceeding and in no way affects the substance of my statements in this Declaration.

II. List of Documents Considered in Formulating My Opinion

8. In formulating my opinion, I have considered the following documents:

Exh. No.	Description
1001	U.S. Patent No. 7,047,526 to Wheeler et al., issued May 16, 2006 ("the '526 patent").
1002	U.S. Patent No. 6,523,172 to Martinez-Guerra, issued February 18, 2003 ("Martinez-Guerra").
1003	Noam Chomsky & George A. Miller, *Finite State Language*, 1 Information and Control 91-112 (1958).
1004	Noam Chomsky, *On Certain Formal Properties of Grammars*, 2 Information and Control 137-167 (1959).
1005	Alfred V. Aho, et. al., Compilers: Principles, Techniques, And Tools (1986).
1006	Gerry Kane, MIPS RISC Architecture (1987).
1007	OpenVMS User's Manual, version 7.1 (Nov. 1996), http://www.mi.infn.it/~calcolo/OpenVMS/ssb71/6489/6489p.htm.

Exh. No.	**Description**
1008	Honeywell Bull, Multics: Commands and Active Functions (1985).
1009	Brian W. Kernighan & Rob Pike, The UNIX Programming Environment (1984).
1010	Cisco's Preliminary Claim Construction Disclosure, *Cisco Systems, Inc. v. Arista Networks, Inc.*, No. 5:14-cv-05344-BLF (N.D. Cal. Aug. 24, 2015).
1011	Robert Sedgewick, Algorithms in C (3rd ed. 1998).
1012	Dennis M. Ritchie & Ken Thompson, The UNIX Time-Sharing System, 17 Communications of the ACM 365 (Jul. 1974).
1013	U.S. Patent No. 6,134,709 to Pratt, issued Oct. 17, 2000.

III. Understanding of Patent Law

9. I have been informed and understand that prior art to the '526 patent includes patents and printed publications in the relevant art that predate the date upon which the alleged invention of the alleged invention of the '526 patent was made.

10. I appreciate that my analysis requires an understanding of the scope of the '526 patent claims to a person having ordinary skill in the relevant art at the time the alleged invention was made ("POSA"). Thus, my analysis of the '526 patent and the prior art is made from the perspective of a POSA at the time of the alleged invention of the '526 patent. In addition, I am aware that patent claims subject to *inter partes* review are given the "broadest reasonable interpretation" as would be understood by a POSA.

11. I have been informed and understand that a claim limitation may be expressed as a means for performing a specified function. I understand that for such "means-plus-function" claim limitations, the claim construction of that limitation must include both the function of the claim limitation and the structure corresponding to that function as described in the specification. I further understand that, where the disclosed structure is a general purpose computer, the specification must also disclose an algorithm for performing the function.

12. I further understand that a claim is invalid if it is anticipated or obvious. Anticipation requires that each and every element of a claim be disclosed expressly or inherently in a single prior art reference.

13. I understand that a claim may be obvious if common sense directs one to modify prior art, combine multiple examples in a prior art reference, or combine multiple prior art references to add missing features to reproduce each and every element of the alleged invention recited in the patent's claims.

14. I also understand that so-called secondary considerations may be relevant to determine whether a claim is obvious should Patent Owner Cisco Systems, Inc. allege such evidence. Such considerations can include evidence of the invention's commercial success, evidence of a long-felt need that was solved by the invention, evidence that others copied the invention, or evidence that the invention achieved a surprising result. I further understand that there must be a nexus or

causal relationship between this evidence and the elements of the claim in order for the consideration to be relevant to the obviousness or non-obviousness of the claim.

IV. Person of Ordinary Skill in the Art

15. I understand that a POSA is one who is presumed to be aware of all pertinent art, thinks along the lines of conventional wisdom in the art, and is a person of ordinary creativity.

16. In my opinion, a POSA at the time of the alleged invention of the ’526 patent would have at least a bachelor’s degree in computer science and 3-5 years of experience in systems development.

V. Technical Background

17. The ’526 patent (Exhibit 1001) discloses “a processor based system having a parser is configured for validating a generic command received from a user relative to a command parse tree” and “issu[ing] a prescribed command for a selected one of the management programs according to the corresponding command format”—*i.e.*, a system that allows a user to enter a generic command which it translates into the format required by a specific management program. Exhibit 1001, Abstract. I understand that the ’526 patent was filed on June 28, 2000. The alleged invention’s claimed benefit is that a user may learn a single generic command language, which the parser-translator converts to a properly formatted com-

mand for a given management tool, thus saving the user from learning the specific syntaxes and formats for multiple management programs. *Id.* at 1:31-37. As the '526 patent explains, "the new syntax provides a generic instruction set that provides an abstraction of the tool-specific command formats and syntax, enabling a user to issue command [sic] based on the relative functions, as opposed to the specific syntax for a corresponding tool 18." *Id.* at 3:31-35; *see also id.* at 7:1-9:16 (listing "Generic Command Examples").

18. Below I provide a brief overview of the relevant technology that one of ordinary skill in the art would have been aware of at the time of the alleged invention.

A. Parsers and parse trees

19. A parser is used to discover the syntactic structure of a phrase or sentence or command, according to the rules of some language. For example, an English language parser might parse the sentence "Ralph bit a brown dog" into a Noun (Ralph) followed by a Verb (bit) followed by a Noun Phrase, which itself contains an Article (a), an Adjective (brown), and a Noun (dog). Once so parsed, the English sentence might then be translated into another language.

20. A computer command line parser might parse the command "watch tcp connections" into a Command Word (watch) followed by a Protocol Type (tcp) followed by a Parameter (connections). Like all parsers, it operates by following

the rules of the language (its "grammar"), which, depending on the complexity of the language, may be expressed as a table, a tree, or some other data structure. English language parsers are quite complex, while computer command line parsers are quite simple. The parsing of computer languages has been studied for decades, and numerous formalisms and methods proposed.

21. Chomsky's foundational work on formal languages in the 1950's proposed a tree representation:

> Every [finite-state, or regular] language can be represented in the form of a *tree*. At the root of the tree is a point from which all sentences start. Each word that can occur as the initial word in some sentence is represented by a branch leaving this initial point. At the end of the branch representing any particular word will be another set of branches representing each of the words that can follow the first. This process of arborization continues until every possible sentence is represented by some path through the tree; or, if the language contains an infinite set of sentences, the tree will continue indefinitely.

Noam Chomsky & George A. Miller, *Finite State Languages*, 1 Information and Control 91 (1958) (Ex. 1003) at 4.

22. One of skill in the art would have known that a language that has a finite number of possible commands is a "regular language" (recognizable by a regular expression or a finite-state machine), which is at Level 3 of Chomsky's well-known hierarchy of languages. *See generally* Noam Chomsky, *On Certain Formal Properties of Grammars*, 2 Information and Control 137 (1959) (Ex.

1004). Such languages can be represented by a finite tree structure, as proposed by Chomsky and Miller (see above).

23. Parsers of regular command languages process commands left-to-right, at each step consulting a data structure such as a table (*see, e.g.*, Alfred V. Aho, et. al., Compilers: Principles, Techniques, and Tool (1986) (Ex. 1005) at 126-27) or a tree (*see, e.g.*, Ex. 1003 at 4) to see what command words are acceptable next choices. As explained, for example in U.S. Patent No. 6,134,709, applied for in June 1998, in its "Background of the Invention" section, "[t]he list of acceptable commands can be stored in a tree structure to reduce the space required to store all of the acceptable commands. The root nodes of the tree can contain acceptable first tokens in a command. . . .The tree can be searched to match the submitted command, token by token." Ex. 1013 at 1:40-57. Methods that use a tree representation of the language would proceed as follows. At the top of the tree (the "root"), where the parse begins, there are one or more branches that lead to nodes, each representing one valid first command word. The nodes may extend to additional branches, leading to other nodes, each representing a subsequent valid command word in light of the word or words that have already been validated. A node that has no branches is called a "leaf."

24. A parser evaluates statements according to a set of syntax rules, known in the art as a "grammar." These syntax rules define valid sequences of

words (sometimes called "tokens") against which the parser compares the incoming statements. A "command parse tree" is one representation of the syntax rules in a command language grammar, but depending on the complexity of the language, such syntax rules can be expressed in other ways, such as a table, as I discussed above. These representations would be obvious to a POSA. For example, below I have listed one way the syntax rules represented in the command parse tree of Figure 2 of the '526 patent can be expressed in the form of rule specifications (a POSA would call these rule specifications "phrase structure rules" (a term originated by Chomsky) or "productions" (*see, e.g.*, Ex. 1005 at 38, 102)):

RULE1 → watch tcp connections

RULE2 → watch udp connections

RULE3 → get tcp connection info

RULE4 → set tcp connection info

These phrase structure rules, or productions, may be associated with some action that the system should perform: "...each rule or *production* of the grammar can be augmented with an action – a statement of what to do when an instance of that grammatical form is found in a program being parsed." Brian W. Kernighan & Rob Pike, The UNIX Programming Environment (1984) (Ex. 1009) at 84.

B. Command translation and translators

25. The notion of translating an input, such as a command expressed in some user-friendly format, into some other format, such as a command for another system component, possibly in a less user-friendly format, using a parser and translator, is fundamental to computer science and engineering. Examples abound:

- *compilers* parse and translate high-level language statements into machine language. Ex. 1005 at 14 ("...a compiler is a program that reads a program written in one language—the *source* language—and translates it into an equivalent program in another language—the *target* language…").

- *assemblers* parse and translate human-readable (and sometimes human-generated) machine level instructions into actual binary machine code. Gerry Kane, MIPS RISC Architecture (1987) (Ex. 1006) at 14 ("The assembler converts assembly language statements into machine code...some MIPS assembly language instructions can generate several R2000 machine instructions.").

- *command languages* parse and translate high-level commands typed by the user into operating system instructions. The Digital Equipment Corporation's *Digital Command Language* or DCL, developed in the 1980's or earlier, was "a set of English-like instructions that tell the operating system to perform specific operations." OpenVMS User's Manual, version 7.1

(Nov. 1996), (available at http://www.mi.infn.it/~calcolo/OpenVMS/ssb71/6489/6489p.htm) (Ex. 1007) at 17, Section 3.2. The manual goes on to say: "Like a spoken language, DCL is made up of *words* (vocabulary) and *word order* (syntax or format)." (*id.* at Section 3.3), and elaborates: "Just as a spoken language depends on the order of words to create meaning, DCL requires that you put the correct elements of the command line in a specific word order or format." *Id.* at 19, Section 3.3.3. An example of a DCL command "shows the general format and parts of a DCL command line:

$ PRINT/COPIES = 5 GROCERY.LIS" *Id.* at 18, Section 3.3.1.

In this example the word "PRINT" is the *name* of the command; "/COPIES" is a *qualifier* that modifies what the command will do; the *value* 5 is the number of copies to print; and "GROCERY.LIS" is the name of the file to print. *Id.* at 18-19, Section 3.3.1.

C. Management Programs

26. The claims of the '526 patent are directed to issuing translated commands to what it calls "management programs." The '526 patent identifies several general examples of management programs, including "Operating Administration and Monitoring (OAM) tools," which are said to be "software-based resources used as administration and/or diagnostic tools for complex processor-based execut-

able software systems, such as software-based unified messaging software systems." Ex. 1001 at 1:11-15. Other examples of OAM tools are provided, including "Real Time Monitoring (RTM) programs" (*id.* at 1:15-16), "Simple Network Management Protocol (SNMP) agents or scripts" (*id.* at 1:25-26), and "external binary files that execute in response to a procedure call" (*id.* at 1:24-25).

27. Such programs had been known in the art for years before the '526 patent. For example, the influential MULTICS system, designed at MIT, had a set of various such programs, each invoked by a command: "A command performs some action for you, such as displaying information on your terminal, formatting a report, or compiling a program." Honeywell Bull, Multics: Commands and Active Functions (Feb. 1985) (Ex. 1008) at 21. A POSA would understand that these actions would be accomplished by different programs. The even more influential Unix operating system had, and has, a set of management programs invoked by commands: "...you can type *commands*, which are requests that the system do something. We will use *program* as a synonym for command." Ex. 1009 at 15. Kernighan and Pike point out that "the simplest command is a single *word*, usually naming a file for execution (later we will see some other types of commands):

```
$ who                              Execute the file /bin/who
you          tty2        Sep 28 07:51
```

```
jpl          tty4       Sep 28 08:32"
```

Id. at 52.

VI. Claim Construction

28. As noted above, I understand that the challenged claims must be given their broadest reasonable interpretation in light of the specification of the '526 patent, which means that the words of the claims should be given the broadest possible meaning, as understood by a person of ordinary skill in the art, consistent with the patent's specification. Below, I offer my opinion as to the broadest reasonable interpretation for some claim elements of the '526 patent.

A. "management program"

29. I understand that in the related district court litigation, Cisco has proposed that "management program" be construed as "separate tools or external agents having their own respective command formats that provide management functions." Cisco Preliminary Claim Construction Disclosure, *Cisco Systems, Inc. v. Arista Networks, Inc.*, No. 5:14-cv-05344-BLF (N.D. Cal. Aug. 24, 2015) (Ex. 1010) at 3. Because this broadly defines a management program to include any program that provides management functions, for example, calls to operating system programs that manage files, report system status, or launch other programs, a POSA would have understood this construction to be the broadest reasonable interpretation of the term "management program."

B. "command parse tree"

30. I understand that Cisco, in related litigation, has proposed construing "command parse tree" as "a hierarchical data representation having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value." Ex. 1010 at 10. The patent specification, along with the illustration in Figure 2 of "command parse tree 22," demonstrates that the patent means to broadly define a command parse tree as a tree representation of the entire language, shown in a hierarchical manner. Therefore, this construction would be understood by a POSA to be the broadest reasonable interpretation of the term "command parse tree."

C. "recursively traversing"

31. A POSA would understand the term "recursively traversing" as used in the '526 Patent to mean "traversing using a process that repeats itself."

32. According to the '526 Patent, the "parser 14 recursively traverses the command parse tree 22 for each command word to identify the best match for the generic command." Ex. 1001 at 3:55-57. Figure 3 of the patent illustrates the process of traversing the command parse tree. *Id.* at Fig. 3; 4:10-11, 4:19-20. In Figure 3, the parser repeatedly executes a particular set of steps. Therefore, a POSA would have understood that under the broadest reasonable interpretation, the term "recursively traversing" means "traversing using a process that repeats itself."

D. "means for validating" / "validating means"

33. I understand this is a "means-plus-function" claim. I further understand that Cisco has proposed the function of this term to be "validating a generic command received from a user." I believe that a POSA would have understood this to be the broadest reasonable interpretation of the term.

34. As described in the patent, "parser 14 is configured for validating a received generic command." *Id.* 3:47-48. The patent further explains that Figure 3 is "a diagram illustrating the method of validating a received generic command" and describes the process for validating the command illustrated by the figure. *Id.* at 3:62-4:36. Thus, a POSA would understand that the specific structure for performing the function "validating a generic command received from a user" is depicted as the parser 14 of Figure 2, described in column 3:36-61, that executes the algorithm disclosed in Figure 3. *Id.*

VII. Prior Art Forming the Basis for Unpatentability

35. U.S. Patent No. 6,523,172 to Martinez-Guerra, *et al.* ("Martinez-Guerra") is entitled "Parser Translator System and Method." Martinez-Guerra (Ex. 1002). I understand that Martinez-Guerra is prior art to the '526 patent under 35 U.S.C. § 102(e) as it was filed on February 19, 1999, claiming priority to a December 1998 provisional application, and issued on February 18, 2003.

36. Martinez-Guerra discloses a parser-translator that allows a user to input a command written in a "high-level user language" and then translates that command into "logically and syntactically correct directives" for specific software tools. Exhibit 1002, Abstract. As in the '526 patent, a benefit of the disclosed parser-translator is that "a user can focus on the semantics of the desired operation and need not be concerned with the proper syntax of language for a particular system." *See id.* Abstract, 10:7-11, 13:30-33.

37. The parser-translator disclosed by Martinez-Guerra includes a token recognizer, a parser, and a translator. *Id.* at 9:24-26. Martinez-Guerra discloses various implementations of the parser-translator, which utilize a data structure representing a language's grammar in the course of performing a translation. *See, e.g.*, 5:51-58, 6:29-44, 18:31-35, Figs. 1-3 (showing "Grammar 17," "Grammar 27," and "Grammar 37"). Although variations in the context and capabilities of external data handling are disclosed, the essential token processing, parsing, and translation functions taught are consistent across Martinez-Guerra's embodiments. *Id.* at 9:6-10 ("FIG. 1 depicts an exemplary data conversion system employing a parser-translator to at least partially define a specific source-to-target data conversion in accordance with a grammar encoding legal statements in a data transformation language."); *id.* at 11:48-51 ("FIG. 3 depicts an illustrative parser-translator component 30 suitable for use in a variety of software tool environments

to parse a stream of tokens in accordance with a grammar . . . ."). Figure 1 of Martinez-Guerra shows how the parser-translator is structured:

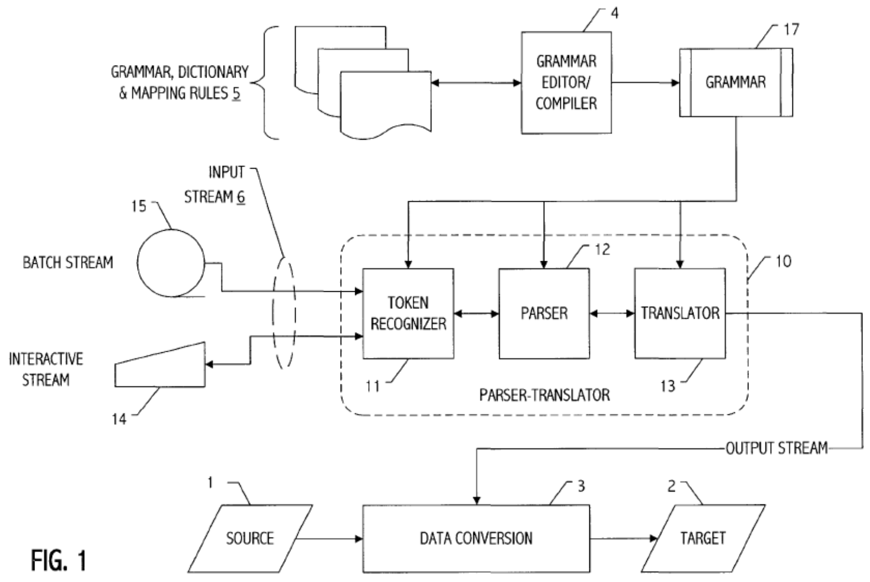


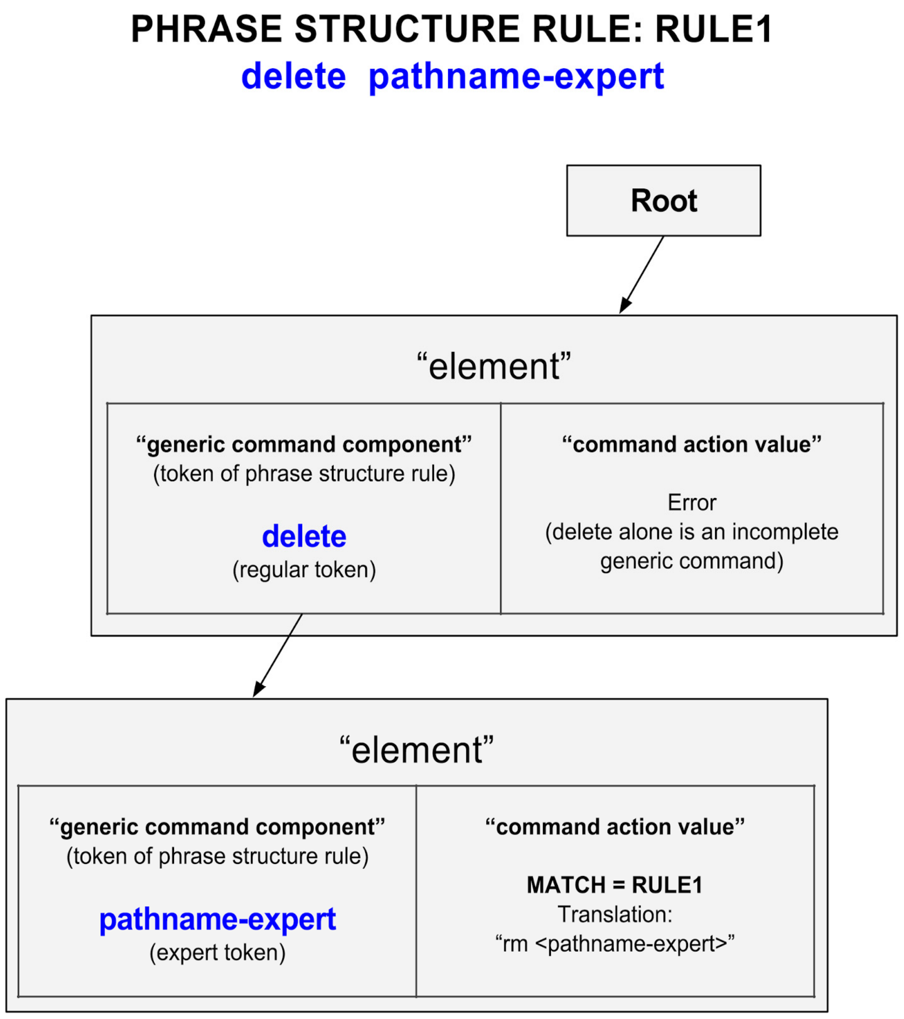
FIG. 1

38. The disclosed parser-translator receives an input "from which a token recognizer 11 extracts tokens in accordance with dictionary entries and rules encoded in grammar 17 and in accordance with a current parse state." *Id.* at 9:36-40. (I note that Martinez-Guerra uses the term "tokens" to describe the input received from the user, while the '526 patent refers to such input as "command words.") Specifically, the parser receives generic commands, such as "delete /usr/extract/testing," from a user via an input stream. *Id.* at 9:35-36, 15:50-62. The translator translates those generic statements into tool-specific outputs based on rules specified in the "grammar." *Id.* at 9:42-44, 3:29-37. The grammar defines valid sequences of words from the input stream through a set of "phrase structure rules," and corresponding translation rules that tell the parser-translator how a val-

id input statement should be translated once a phrase structure rule is satisfied. *Id.* at 9:40-45, 7:52-64. Martinez-Guerra points out that "a variety of suitable representations and encodings of a grammar are suitable . . . .[P]ersons of ordinary skill in the art will appreciate a variety of alternative representations and encodings." *Id.* at 13:59-65. A POSA would understand that a *tree* structure is one such suitable representation, if the language described by the grammar is *finite* (has only a finite number of possible valid statements). A POSA would also understand that finite command languages constitute a subset of the languages addressed by Martinez-Guerra, which describes translating commands for a variety of tools, just as a POSA would understand that the command languages addressed by the '526 patent are finite. The grammar also uses dictionary entries that contain all of the input words that the parser-translator should recognize. *Id.* at 14:16-18. Dictionary entries correlate input words with grammar categories. *Id.* at 7:36-37. Some tokens in the dictionary are called "expert tokens"—they denote a grammar category of acceptable user inputs rather than a specific one. A *filename* path would be an example of such a category; an *even number* would be another. *Id.* at 7:38-44, 15:6-12.

39. As I explain in greater detail below, for any finite command language, Martinez-Guerra's phrase structure rules constitute a command parse tree because they are collectively a hierarchical data representation of the valid sequences of

components of statements in a high-level language, and because every component of each phrase structure rule has an action that will occur if that component is the last valid component parsed. The phrase structure rule RULE1, which I use as an example, would be represented as a set of component-action value pairs (what the ’526 patent calls “elements”) within such a command parse tree, as I have illustrated in the figure below:


PHRASE STRUCTURE RULE: RULE1
delete pathname-expert
Root
“element”
“generic command component”
(token of phrase structure rule)
delete
(regular token)
“command action value”
Error
(delete alone is an incomplete generic command)
“element”
“generic command component”
(token of phrase structure rule)
pathname-expert
(expert token)
“command action value”
MATCH = RULE1
Translation:
“rm <pathname-expert>”

40. As the input stream is processed, the parser-translator uses the phrase structure rules, which, as illustrated in the figure above, constitute a command parse tree, to determine valid ("legal") next parse states—*i.e.*, the tokens that, as defined by the grammar, may follow the token(s) already received. *Id.* at 10:42-58, 18:31-48. The parser "builds an internal representation of the translation of the input stream according to translation rules encoded in the grammar." *Id.* at 10:50-53. When a valid sequence of received tokens uniquely matches a single phrase structure rule, the parser-translator can carry out the translation corresponding to that rule. *Id.* at 10:56-58, 20:45-49. Once translated, the output statement (*e.g.*, a prescribed command) for a given software tool is issued and can thereafter be executed. *See, e.g.*, *id.* at 15:50-62.

41. Throughout my analysis of the '526 patent claims, I use Martinez-Guerra's discussion at column 15:50-16:5 as a working example of how its teachings disclose the relevant limitations. This discussion explains how the parser-translator would translate a generic command received from a user ("delete /usr/extract/testing") into the corresponding prescribed command ("rm /usr/extract/testing") using the phrase structure rule "RULE1" in the mar. RULE1 describes the legal ordering of tokens ("delete pathname-expert") that will satisfy the rule. Once the rule is satisfied, the parser-translator will perform the translation from the generic command to the prescribed command. *Id.* at

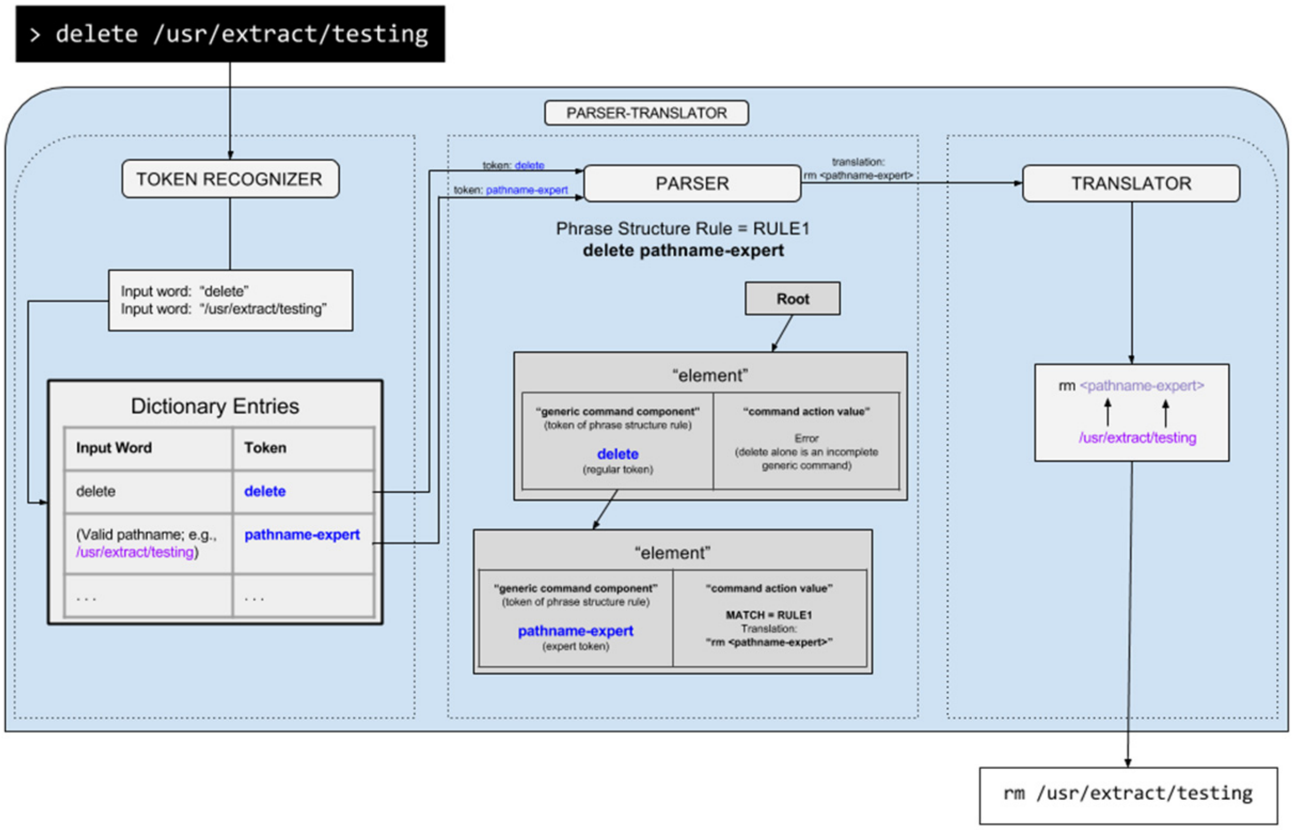
8:24-26, 14:10-11, 15:54-59. The tokens that are compared to those in RULE1 are determined by matching each input word of the generic command received from the user ("delete /usr/extract/testing") to a corresponding token found in the dictionary entries of the grammar. *Id. at* 9:34-38, 14:60-15:15. Martinez-Guerra explains that the dictionary entries identify all of the tokens that the parser-translator should recognize, and acts as a table for correlating an input command word with a corresponding token (here I use "token" in the sense of the '526 patent). *Id.* at 14:16-18, 14:60-62, 20:22-24.

42. In this example, the generic command includes two command words, "delete" and "/usr/extract/testing." Before the command words are compared with RULE1, each word is checked to confirm that it is present in the dictionary entries, which confirm that the input words are valid for use with the parser-translator. *Id.* at 9:34-38, 15:54-16:5. "[D]elete" is a "regular token" because it is a fixed sequence of characters, and the content of the token would be the same as the input word ("delete"). *Id.* at 15:2-4, 15:58-59. The input word "/usr/extract/testing" is what Martinez-Guerra calls an "expert token," which is "a string whose value cannot be specified when the grammar is written but which can be defined in terms of its characteristics." *Id.* at 15:4-6, 15:67-16:5. The expert token "receives its value from the user, and the grammar imposes restrictions on the type of value accepted." *Id.* at 15:6-8. The input word "/usr/extract/testing" specifies the pathname of

a particular file that the user wishes to operate upon. Because there are countless files that a user could identify for this command word, Martinez-Guerra explains that the dictionary entries include validation functions for determining that the word "/usr/extract/testing" is a pathname, as opposed to a regular token. This input word corresponds to the token "pathname-expert." *Id.* at 15:6-15, 17:10-24, 20:10-28. The parser-translator therefore matches "/usr/extract/testing" with the token "pathname-expert" in the dictionary entries. Thereafter, when determining the presence of tokens in elements of the phrase structure rules, the parser-translator will look for the token "pathname-expert" in a rule in the place of the input word "/usr/extract/testing." *Id.* at 15:54-16:5.

43. If the sequence of tokens matches a sequence in a phrase structure rule, then that rule is satisfied, and a corresponding translation occurs. *Id.* at 9:40-43, 20:45-48. Thus, because the tokens "delete" and "pathname-expert" satisfy RULE1, the translator is instructed to perform the translation as "rm <pathname-expert>." *Id.* at 15:59-62. Martinez-Guerra explains that, in this example, "<path-name-expert>" will be replaced with the original input word, thus resulting in the translator issuing the prescribed command "rm /usr/extract/testing." *Id.* at 15:65-16:5.

44. I have prepared the following graphic which illustrates the working example from column 15:50-16:5, as discussed above.


> delete /usr/extract/testing
PARSER-TRANSLATOR
TOKEN RECOGNIZER
Input word: "delete"
Input word: "/usr/extract/testing"
Dictionary Entries
Input Word | Token
delete | delete
(Valid pathname; e.g., /usr/extract/testing) | pathname-expert
. . . | . . .
token: delete
token: pathname-expert
PARSER
translation: rm <pathname-expert>
Phrase Structure Rule = RULE1
delete pathname-expert
Root
"element"
"generic command component" (token of phrase structure rule)
delete (regular token)
"command action value"
Error (delete alone is an incomplete generic command)
"element"
"generic command component" (token of phrase structure rule)
pathname-expert (expert token)
"command action value"
MATCH = RULE1 Translation: "rm <pathname-expert>"
TRANSLATOR
rm <pathname-expert>
/usr/extract/testing
rm /usr/extract/testing

45. Martinez-Guerra teaches that, as each word in a command is parsed, the phrase structure rules define a set of valid next tokens in light of the tokens that have already been validated. *See, e.g., id.* at 18:36-40 ("Token recognizer 31 computes a set of valid next states . . . [that] includes the set of next tokens consistent with a current parse state.") At first, the set of valid tokens includes all "tokens that may begin a legal statement in the language defined by the grammar." *Id.* at 18:40-42. Each successive token is then compared to the set of valid next tokens in light of the tokens that came before. *See, e.g., id.* at 20:31-34 ("Token recognizer 31

performs input matching against each of the valid next states corresponding to a parse state representation consistent with the input stream read so far.")

VIII. Grounds for Invalidity of the '526 Patent

46. I understand that the asserted grounds for the unpatentability of the '526 patent are that claims 1-26 are rendered obvious by Martinez-Guerra. Below I demonstrate how each of the claim elements in the '526 patent is disclosed or rendered obvious by Martinez-Guerra.

A. Claim 1

1. [1A] "method in a processor-based system configured for executing a plurality of management programs according to respective command formats, the method comprising"

47. Martinez-Guerra discloses a parser-translator that uses a high-level user language to interface with a plurality of software applications. *Id.* at 3:48-52. The parser-translator translates statements from a high-level language, which can include generic commands, to "logically and syntactically correct directives for performing the desired data transformations or operations," *e.g.*, prescribed commands in the proper format, using one or more software tools. *Id.* at 3:29-37. The method disclosed by Martinez-Guerra is carried out in software executed on a computer system. *Id.* 9:10-13.

48. The broad disclosure of software tools that can be used with the system includes "management programs" within the meaning of the '526 patent. For example, Martinez-Guerra explains that an "illustrative enterprise tools environ-

ment includ[es] data discovery and cleansing tools, data extraction conversion and migration tools, data movement and replication tools, query Multi-Dimensional Data (MDD) analysis and On-Line Analytical Processing (OLAP) tools, as well as applications and gateways that may advantageously incorporate a parser-translator component." *Id.* at 11:36-42. Martinez-Guerra further explains that the disclosed parser-translator is suitable for software applications such as "query and on-line analytical processing tools, gateways to operational data systems" and "enterprise resource planning applications." *Id*. at 13:17-20.

49. The disclosed tools are separate from the parser-translator and each other, and use a command format that may differ from the high-level user language command provided by the user in the input stream. *See id.* at 15:50-62. As illustrated by Martinez-Guerra's example in column 15, for example, the high-level input command "delete /usr/extract/testing" is translated for a UNIX environment to the prescribed UNIX command "rm /usr/extract/testing." *Id.* Indeed, Martinez-Guerra states that a benefit of the disclosed parser-translator is that "a user can focus on the semantics of the desired operation and need not be concerned with the proper syntax of language for a particular system." *See id.* Abstract, 10:7-11, 13:30-33.

2. [1B] "receiving a generic command from the user;"

50. Martinez-Guerra discloses a user interface that receives from a human user an input stream consisting of statements in a "high-level user language[.]" *Id.* at Abstract; *see also* Fig. 1, 9:15-23. Because a statement in a "high-level user language," as described in Martinez-Guerra, is an abstraction of a desired operation that is later translated into a directive for a specific software tool, one of ordinary skill in the art would understand that such a statement can be a "generic command" within the meaning of the '526 patent. Ex. 1002 at 10:7-11, 13:30-33.

51. For example, Martinez-Guerra discloses that the parser-translator may receive an input stream including the command "delete /usr/extract/testing." *Id.* at 15:50-16:5 ("the string delete in an input stream such as input stream 36"). One of ordinary skill in the art would understand that the command "delete /usr/extract/testing" is analogous to the "Generic Command Examples" disclosed in the '526 patent. *See* Ex. 1001 at 7:1-9:15 (identifying *e.g*, "set watchtime <milli-seconds>" as an exemplary generic command). As explained in the limitations that follow, the generic "delete /usr/extract/testing" command received in the input stream is validated by the parser-translator and ultimately translated into the prescribed UNIX "rm /usr/extract/testing" command according to a translation function encoded in the grammar.

3. [1C.1]: "validating the generic command based on a command parse tree that specifies valid generic commands relative to a prescribed generic command format,"

52. Martinez-Guerra discloses the use of a command parse tree as claimed in the '526 patent. Martinez-Guerra discloses that words in the input stream (which Martinez-Guerra refers to as "tokens") are analyzed by the token recognizer subcomponent of the parser-translator according to dictionary entries and the rules encoded in the grammar. *Id.* at 10:35-38. Martinez-Guerra defines "grammar" as "a formal definition of the syntactic structure of a language typically represented as a set of rules that specify legal orderings of constituents and subconstituents (*e.g.*, phrases and symbols) in a language." *Id.* at 7:52-56. The "legal orderings" refer to valid sequences of tokens as defined by phrase structure rules in the grammar. *Id.* at 8:24-26. The grammar also provides translation functions corresponding to all such valid sequences of tokens. *Id.* at 16:15-16, 17:51-52.

53. Martinez-Guerra explains that the parser-translator creates internal data structures, including "parse state data structures" and "internal representations of a grammar," which it uses to keep track of next legal choices in the grammar. *Id.* at 18:31-40, 20:31-34, and 10:42-49. "[T]he set of valid next states includes the set of next tokens consistent with the current parse state." *Id.* at 18:38-40. Stated differently, the parser-translator validates the words in the user input stream using the phrase structure rules, which define valid next tokens in light of the tokens that

have already been validated. In the case of rules such as RULE1 (and those rules illustrated in Figure 2 of the '526 patent), phrase structure rules together constitute a command parse tree, which specifies the possible valid next legal choices until the input stream has been fully parsed. *Id.* at 10:42-58, 20:45-49. A person of ordinary skill in the art would understand that for finite command languages, Martinez-Guerra's phrase structure rules constitute a command parse tree because they are collectively a hierarchical data representation of the valid components of statements in a high-level command language, and because, as I demonstrate below, every component of each phrase structure rule has an action that will occur if that component is the last valid component parsed.

54. When the input stream contains a generic command, as in the "delete" example (*id.* at 15:50-62), the command is validated by the parser-translator relative to a prescribed generic command format defined by phrase structure rules in the grammar. *Id.* at 10:42-58. Specifically, the token recognizer subcomponent receives the "delete" command and first determines whether the word "delete" matches a token in the dictionary entries, which define all tokens found in the phrase structure rules of the grammar. *Id.* at 9:34-38, 14:16-18. Once it has confirmed a match, the token recognizer then determines whether the token (in the '526 sense) corresponding to the "delete" is present among the next legal parse states maintained by the parser. *Id.* at 9:34-40, 20:28-34.

55. As discussed above, each parse state corresponds to the parser's arrival at a node in the command parse tree, which shows all valid next legal tokens, as defined by the phrase structure rules. In the example found in column 15, the tokens "delete" and "pathname-expert" are matched, respectively, with the first and second words of the phrase structure rule, "RULE1." *Id.* at 15:50-62. Because a phrase with the tokens "delete" and "pathname-expert," in that order, matches the phrase structure rule RULE1—*i.e.*, the "prescribed generic command format"—the generic command is validated.

4. [1C.2] "the command parse tree having elements each specifying at least one corresponding generic command component and a corresponding at least one action value,"

56. As discussed above, Martinez-Guerra discloses that the grammar includes "a set of phrase structure rules and associated translation rules that define the syntactic order of a source and target language . . . .". *Id.* at 7:57-59, 17:51-52. A person of ordinary skill in the art would understand that for finite command languages, Martinez-Guerra's phrase structure rules constitute a command parse tree because they are collectively a hierarchical data representation of the valid components of statements in a high-level command language, and because, as I discuss below, every component of each phrase structure rule has an action that will occur if that component is the last valid component parsed. Phrase structure rules like "RULE1" together define legal orderings of tokens according to a partic-

ular language, such as a high-level language (*id.* at 14:11-12), and can be represented as a set of textual rule specifications (*e.g.*, "RULE1 → delete pathname-expert," *supra* at ¶ 24). In the textual rule specification for RULE1, each token to the right of the arrow is a component needed to form a complete statement; each of these is what the '526 patent calls a "generic command component."

57. Corresponding to each of these components of the phrase structure rule is an appropriate action that will occur if there are no further words to validate against the command parse tree (if that component is a "best match" for the generic command). If an input word is invalid or a command is incomplete—for example, if only the word "delete" is received without any pathname to define the target of the operation—that action may result in returning an error. *See id.* at 19:3-4. On the other hand, a valid input stream that matches and completes a phrase structure rule—*e.g.*, "delete /usr/extract/testing" (*id.* at 16:5)—will correspond to performing the translation associated with the phrase structure rule RULE1. *Id.* at 8:60-65. The instruction to return an error that results from inputting an invalid word or an incomplete sequence (*e.g.*, inputting "delete" without a target pathname), or the translation function that results from inputting a complete and valid sequence of tokens (*e.g.*, "delete /usr/extract/testing"), is what the '526 patent refers to as a "command action value." Such a command action value exists for each generic command component of a given phrase structure rule because, as input words are

processed, they may be invalid, resulting in an error, or valid, leading ultimately to the satisfaction of a phrase structure rule, thus resulting in a translation according to the corresponding translation function.

58. Thus, in Martinez-Guerra, each legal choice according to a phrase structure rule is an "element," as described in the '526 patent, which has both a generic command component and a corresponding command action value.

5. [1C.3] "the validating step including identifying one of the elements as a best match relative to the generic command; and"

59. In the generic command example discussed above, Martinez-Guerra explains that the generic command "delete /usr/extract/testing" is validated by the parser-translator because it matches the legal ordering of tokens described by the phrase structure rule RULE1. *Id.* at 15:50-62. The element corresponding to this final token of RULE1, "pathname-expert," is the "best match" relative to the generic command because it is the final thing necessary to satisfy the phrase structure rule, thus causing the parser-translator to translate the input stream into a prescribed command. As Martinez-Guerra states, "[o]nce the current parse state includes a complete phrase as defined by phrase structure rules encoded in grammar 37, translator 33 provides a translation." *Id.* at 11:57-60.

6. [1D] "issuing a prescribed command of a selected one of the management programs according to the corresponding command format, based on the identified one element."

60. Martinez-Guerra discloses that when a phrase structure rule is satisfied by the input stream, the translation function associated with the phrase structure rule causes the parser-translator to perform the translation. *Id.* at 11:55-62, 14:10-21, 17:47-52. The translator outputs the translation in a format for a software tool. *Id.* at 13:29-33. As Martinez-Guerra explains, "the translation functionality of parser-translator component 30 provides the capability of translating the now properly formed user language statements *to the form and syntax desired by a specific software tool*." *Id.* (emphasis added).

61. In the case of the generic "delete /usr/extract/testing" command, after matching the final token with a next legal choice in the parse state, Martinez-Guerra explains that the parser-translator carries out the translation using the translation function associated with phrase structure rule "RULE1," resulting in the prescribed UNIX command "rm /usr/extract/testing." *Id.* at 15:59-62, 17:51-52. One of ordinary skill in the art would recognize that "rm" is the UNIX shell command used to delete files in a computer system. Martinez-Guerra thus discloses issuing a prescribed command for removing a file in a selected one of the management programs according to the corresponding command format for that program ("rm /usr/extract/testing"). *Id.* at 15:50-62, 16:5.

B. Claim 2

1. [2A] "The method of claim 1, wherein the generic command includes at least one input command word, the validating step including:"

62. Martinez-Guerra discloses a generic command "delete /usr/extract/testing," which includes at least one input command word, "delete." *Id.* at 15:50-62. As explained, "delete" in RULE1 is a token "matching the string delete in the input stream such as input stream 36." *Id.*

2. [2B] "comparing each input command word to a command word translation table, configured for storing for each prescribed command word a corresponding token, for identification of a matching token; and"

63. The token recognizer subcomponent of the parser-translator extracts input command words from the input stream "in accordance with dictionary entries and rules encoded in grammar 17 and in accordance with a current parse state." *Id.* at 9:35-38, 11:51-55. Martinez-Guerra explains that dictionary entries "encode information associated with tokens" (*id.* at 10:24-25), and "describe all the tokens that a parser-translator (*e.g.*, token recognizer 31 of parser-translator component 30) should recognize . . . ." *Id.* at 14:16-18. The '526 patent describes the "command word translation table" in similar terms: it "includes all the command words 26 that are valid according to the generic syntax." Exhibit 1001 at 3:45-46. One of ordinary skill in the art would understand from these disclosures, and in view of Martinez-Guerra's example of the "delete /usr/extract/testing" generic command,

that the dictionary entries in Martinez-Guerra are a translation table used by the token recognizer to match input words with valid tokens found in the phrase structure rules (*e.g.*, "delete" in the exemplary "RULE1"). *See* Exhibit 1002 at 14:55-59. As Martinez-Guerra explains, "[a] rule specification includes a sequence of grammar categories that define a legal sequence (or sequences) of tokens in the language. Tokens (read from the input stream) 'match' categories." *Id.*

64. For example, when the user submits the "delete /usr/extract/testing" command, the first input word "delete" is matched with a "regular token" entry in the dictionary having the same content ("delete"). As Martinez-Guerra explains, a "regular token is a fixed sequence of characters (for example, the string 'delete')." *Id.* at 15:2-4. The next (and final) input word the user submits in this example is the pathname for the file to be deleted ("/usr/extract/testing"). *Id.* at 15:57-16:5. This input word is what Martinez-Guerra refers to as an "expert token," and the parser-translator uses the dictionary entries to match that input word with the token "pathname-expert." *Id.* at 15:4-15, 15:63-16:5. The parser-translator then uses the matching token "pathname-expert" from the dictionary entries instead of the original input word "/usr/extract/testing" when searching for a match in the command parse tree.

3. [2C] "determining a presence of the matching token within the command parse tree for each input command word."

65. For each input command word that the token recognizer subcomponent matches with a corresponding token in the dictionary (*id.* at 11:51-55), it determines whether that token is among the next legal choices in the parse states maintained by the parser. *Id.* at 18:36-42, 18:66-19:7, 20:31-34, and 10:42-49. A person of ordinary skill in the art would understand that for any finite command language, Martinez-Guerra's phrase structure rules constitute a command parse tree because they are collectively a hierarchical data representation of the valid components of statements in a high-level command language, and because, as I discussed above, every component of each phrase structure rule has an action that will occur if that component is the last valid component parsed. *See supra* at ¶ 30; *see also id.* at 14:10-15, 18:38-48. If the parser-translator determines that an input token matches an expected next legal choice in the parse state, then it adds the token to the parse state, and the parser computes an updated list of potentially valid next states. *Id.* at 19:5-35. Stated in terms of the '526 patent, the parser-translator determines the presence of a matching token (*e.g.*, "delete") within the command parse tree.

66. In the example discussed in the previous limitation, the parser-translator first determines the presence of the token corresponding to the input word "delete" in the command parse tree. *Id.* at 15:50-62, 18:31-48. A matching

token is found within the command parse tree because "delete" is the first token in the phrase structure rule RULE1. *Id.* at 15:57; 20:40-45, Fig. 10. Having matched the first token, the parser-translator then determines the presence of the second (and final) token, which Martinez-Guerra explains is the token "pathname-expert," corresponding to a valid file path name supplied by the user ("/usr/extract/testing"), as explained in the previous limitation. *Id.* at 15:50-16:5, 20:40-45. The presence of "pathname-expert" would also be found in the command parse tree because it corresponds to the final token of the phrase structure rule RULE1. *Id.* at 15:57. The parser-translator of Martinez-Guerra thus determines the presence of a matching token in the command parse tree for each input command word.

C. Claim 3

1. "The method of claim 2, wherein the determining step includes recursively traversing the command parse tree based on an order of the input command words for identification of the matching token within the identified one element."

67. As discussed above, the Martinez-Guerra token recognizer determines, for each input command word, the presence of a matching token in the phrase structure rules of the grammar, which together constitute a command parse tree as described in the '526 patent. *Id.* at 18:31-40. Martinez-Guerra teaches that "[t]oken recognizer 31 performs input matching against each of the valid next states corresponding to a parse state representation consistent with the input stream

read so far." *Id.* at 20:31-34. As successive tokens are matched to potential next legal choices (*i.e.*, "elements") in the representation of the current parse state (*id.* at 20:35-40), the parser-translator updates the current parse state. *Id.* at 20:42-45, 20:49-54. This process of matching input tokens to next legal choices in the phrase structure rules is repeated for each word in the input stream until a phrase structure rule indicates that the input stream has been fully parsed. *Id.* at 20:42-48. Martinez-Guerra thus teaches processing the input stream based on an order of the input command words, namely, the left-to-right order.

68. One of ordinary skill in the art would understand that this process "recursively traverses" the command parse tree in the same way as the '526 patent, using the broadest reasonable construction of "recursively traverses," namely "traversing using a process that repeats itself." *See supra* at ¶ 32. Figure 10 of Martinez-Guerra "illustrates the application of an input token to each parse state" (*id.* at 20:40-41), and this process repeats for each input token in the order received. *Id.* at 10:53-55, 18:62-65. "Additional tokens are parsed from the input stream (*e.g.*, input stream 36) until a complete statement is defined in accordance with the particular grammar employed." *Id.* The same process is used in Figure 3 of the '526 patent, where tokens are iteratively matched to elements in the command parse tree. Because Martinez-Guerra uses the same process for matching to-

kens to next legal choices in the command parse tree as the '526 patent, it "recursively traverses" the command parse tree as claimed by the '526 patent.

69. To the extent that the Board were to conclude that "recursively traversing" within the meaning of the claim requires the use of a recursive process in the mathematical sense (i.e., one that calls itself), the use of such a process would be obvious over Martinez-Guerra in light of the understanding of a POSA. A POSA would understand that repeating processes and recursive processes were well-known alternatives for performing tasks such as those described in claim 3, and that one could substitute a recursive process for a repeating process to achieve the same predictable results. Indeed, recursive processing (in the stricter sense) has long been seen as particularly well-suited to trees: "The study of recursion is intertwined with the study of recursively defined structures known as trees. . . . A full discussion of recursion and trees could fill an entire book, for they arise in many applications throughout computer science . . . ." Robert Sedgewick, Algorithms in C (3rd ed. 1998) (Ex. 1011) at 3-4.

D. Claim 4

1. "The method of claim 3, wherein the issuing step includes issuing the prescribed command based on a corresponding command key specified for the matching token within the identified one element."

70. Martinez-Guerra discloses issuing a prescribed command based on a "translation function identifier" specified for the element that completes a phrase

structure rule, such as the "pathname-expert" element of "RULE1." *Id.* at 15:50-62, 16:11-16. The "translation function identifier" specifies where a translation function corresponding to a matched phrase structure rule can be located. *Id.* at 16:11-16, 20:61-64. It is thus a "command key," as claimed in the '526 tent. Martinez-Guerra explains that the translation function identifier "may take the form of . . . an index specifying a function number," and "allows parser-translator component 30 (particularly token recognizer 31 and translator 33 sub-components thereof) to invoke the corresponding . . . translation function in accordance with the grammar 37." *Id.* at 16:16-24. When the input stream contains a legal ordering of tokens according to a phrase structure rule, the translation function corresponding to that phrase structure rule is used to determine the translation. *Id.* at 17:47-52, 20:61-64. The output of the translation process is the "prescribed command."

71. Thus, in the example discussed in claim 1 above, when the final token, "pathname-expert" (which is used in place of the input word "/usr/extract/testing" in the generic command "delete /usr/extract/testing," received from the user) is matched with the "pathname-expert" element of the command parse tree containing "RULE1," a translation function identifier specified for RULE1 tells the parser-translator where to locate the translation function needed for the translator to issue the prescribed command "rm /usr/extract/testing." Ac-

cordingly, the prescribed command is based on a corresponding command key specified for the matching token "pathname-expert" within the identified one element.

E. Claim 5

1. "The method of claim 4, wherein the issuing step further includes accessing a prescribed translator configured for converting the generic command according to the corresponding command format into the prescribed command based on the corresponding command key."

72. Martinez-Guerra discloses accessing a prescribed translator (*e.g.*, translator 33) that is configured for converting the generic command according to the corresponding command format (*e.g.*, "delete /usr/extract/testing") into the prescribed command (*e.g.*, "rm /usr/extract/testing") based on the corresponding command key (*e.g.*, translation function identifier corresponding to phrase structure rule RULE1). "A translation function is associated with a phrase structure rule and is invoked by translator 33." *Id.* at 17:51-58. For example, the translation from the generic command "delete /usr/extract/testing" to the prescribed command "rm /usr/extract/testing" is based on the translation function corresponding to phrase structure rule RULE1. *Id.* at 15:50-62. When a phrase structure rule, such as RULE1, is satisfied by the input words received, Martinez-Guerra explains that the translation corresponding to that rule is performed by a translator component of the parser-translator. *Id.* at 13:29-33, 15:24-28, 17:51-58, 20:61-64. In carrying out

such a translation, the prescribed translator for converting the generic command to the prescribed command is necessarily "accessed." As discussed above with reference to claim 4, the accessed translator performs this conversion based on the translation function identifier, which functions as a "command key."

F. Claim 6

1. "The method of claim 5, wherein the validating step including validating at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command, the issuing step including issuing the prescribed command based on the identified one element corresponding to the portion of the generic command."

73. As discussed above with reference to claim 1, the parser-translator validates the generic command "delete /usr/extract/testing" because the token recognizer determines, for each of the input words "delete" and "/usr/extract/testing" (where "/usr/extract/testing" is matched with the token "pathname-expert" in a dictionary entry), that the token corresponding to each input word is a legal choice according to the command parse tree representation of the phrase structure rule RULE1. *See id.* at 15:50-62, 18:36-42, 18:66-19:7, 20:31-34, and 10:42-49. By matching the final token ("pathname-expert") to the corresponding element in the command parse tree representation of RULE1 ("pathname-expert"), Martinez-Guerra discloses validating at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic

command. Martinez-Guerra satisfies this limitation because validating the entire generic command "delete /usr/extract/testing" necessarily validates "at least a portion of" the generic command. Further, Martinez-Guerra discloses that if an input word is invalid or a command is incomplete—for example, if only the word "delete" is received without any pathname to define the target of the operation—that action may result in returning an error. *See id.* at 19:3-4. In this circumstance, the parser-translator identifies a "best match" for less than the entire command.

74. Martinez-Guerra further issues the prescribed command "rm /usr/extract/testing" based on the identified one element of the command parse tree ("pathname-expert") corresponding to the portion of the generic command ("/usr/extract/testing"). As discussed above with reference to claim 1, "pathname-expert" is the final element in the command parse tree representation of the phrase structure rule RULE1. Martinez-Guerra explains that once the final token in the sequence of legal tokens for a phrase structure rule is matched, the translation function corresponding to the phrase structure rule is used by the translator to output a translation. *Id.* at 9:40-44, 15:24-28, 17:47-52, 20:61-64. Accordingly, when the "pathname-expert" token (which takes the place of the input word "/usr/extract/testing" in the generic command) is matched with the final element "pathname-expert" of RULE1, the input command is fully parsed, and the translation function corresponding to RULE1 is used to issue the prescribed command

"rm /usr/extract/testing." *Id.* at 20:61-64, 21:1-4. Martinez-Guerra explains that this is accomplished by the translator substituting the original input word ("/usr/extract/testing") for the token "pathname-expert" as part of the translation function ("rm <pathname-expert>"). *Id.* at 15:65-16:5. The issued prescribed command is thus "rm /usr/extract/testing." *Id.* at 16:5.

G. Claim 7

1. "The method of claim 6, further comprising executing the prescribed command within the corresponding selected one management program."

75. As discussed with reference to claim 1, Martinez-Guerra discloses translating the generic command "delete /usr/extract/testing" into the prescribed command "rm /usr/extract/testing," which is an executable command for deleting the file at "/usr/extract/testing" using a UNIX shell program. *Id.* at 15:50-62. Martinez-Guerra's "delete" example focuses on the translation capability of the parser-translator and, accordingly, does not expressly state that the UNIX shell program then executes the resulting command. One of ordinary skill in the art would understand from Martinez-Guerra's disclosure that the purpose of such a translation is for the UNIX shell program to execute—or "perform[] the desired data transformation[] or operation[]," *id.* at 3:36-37—according to the prescribed command. In light of Martinez-Guerra's teachings, it would thus be obvious that the UNIX shell

program would thereafter execute the prescribed "rm /usr/extract/testing" command.

76. More generally, a POSA would understand that generic commands in Martinez-Guerra are translated so that they may be executed. Martinez-Guerra states that an objective of the invention is to allow users to "specify complex . . . transformation statements in a high-level user language, . . . and to translate [those statements] into logically and syntactically correct directives *for performing the desired data transformations or operations.*" *Id.* at 3:30-37 (emphasis ed). One of ordinary skill in the art would understand from this disclosure that Martinez-Guerra intended that the translated directives ("prescribed commands") were intended to be executed by the software tools corresponding to those directives. Indeed, a translated directive must be executed in order to "perform the desired data transformation or operation." Further, if the directives were not executed, there would be no purpose in carrying out the translation to begin with.

H. Claim 8

1. "The method of claim 1, wherein the validating step including validating at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command, the issuing step including issuing the prescribed command based on the identified one element corresponding to the portion of the generic command."

77. Claim 8 contains limitations that are identical to claim 6, but it depends from claim 1 rather than from claim 5. Because it includes no additional limitations other than those found in claim 6, the elements of claim 8 are disclosed by Martinez-Guerra as described above for claim 6.

I. Claim 9

1. "The method of claim 8, further comprising executing the prescribed command within the corresponding selected one management program."

78. Claim 9 contains limitations that are identical to claim 7, but it depends from claim 1 rather than from claim 6. Because it includes no additional limitations other than those found in claim 7, the elements of claim 9 are disclosed by Martinez-Guerra as described above for claim 7.

J. Claim 10

1. [10A] "A system configured for executing a plurality of management programs according to respective command formats, the system comprising:"

79. Martinez-Guerra discloses the claimed system, wherein a parser-translator uses a high-level user language to interface with a plurality of software

applications. *Id.* at 3:48-52. It does so by translating statements from a high-level language which can include generic commands, to "logically and syntactically correct directives for performing the desired data transformations or operations" *e.g.*, prescribed commands in the proper format, using one or more software tools. *Id.* at 3:29-37. The method disclosed by Martinez-Guerra is carried out by software executed on a computer system. *Id.* at 9:10-13.

80. The broad disclosure of software tools that can be used with the system includes "management programs" within the meaning of the '526 patent. For example, Martinez-Guerra explains that an "illustrative enterprise tools environment includ[es] data discovery and cleansing tools, data extraction conversion and migration tools, data movement and replication tools, query Multi-Dimensional Data (MDD) analysis and On-Line Analytical Processing (OLAP) tools, as well as applications and gateways that may advantageously incorporate a parser-translator component." *Id.* at 11:36-47. Moreover, additional software applications include "query and on-line analytical processing tools, gateways to operational data systems." *Id.* at 13:14-20. The disclosed tools are separate from the parser-translator and may each have their own respective command formats. As illustrated by the example in Martinez-Guerra discussed above, the relevant tool may use a command format that differs from the high-level language command provided by the user in the input stream. *See id.* at 15:50-62 (translating the high-level language

command "delete /usr/extract/testing" to the prescribed UNIX command "rm /usr/extract/testing").

2. [10B.1]: "a parser having a command parse tree configured for validating a generic command received from a user,"

81. As described above with respect to claim 1, the parser-translator of Martinez-Guerra, which includes a parser subcomponent (*id.* at 8:18-23, 9:24-26, Figs. 1-3), is configured to validate generic commands received from the user using phrase structure rules. A person of ordinary skill in the art would understand that for any finite command language, Martinez-Guerra's phrase structure rules constitute a command parse tree because they are collectively a hierarchical data representation of the valid components of statements in a high-level command language, and because every component of each phrase structure rule has an action that will occur if that component is the last valid component parsed. Martinez-Guerra discloses a user interface that receives from a user an input stream in a high-level language. *Id.* at Fig. 1; 4:4-9; 9:15-23. The parser-translator translates statements in that language into "statements or directives appropriate to a particular data processing application." *Id.* at 4:4-9. One of ordinary skill in the art would understand that the high-level language described in Martinez-Guerra includes "generic commands" within the meaning of the '526 patent, because the input can be an abstraction of a desired operation that is later translated into a directive for a specific software tool. *See id.* at 10:7-11, 13:30-33. For example, Martinez-

Guerra discloses that the parser-translator may receive an input stream including the word "delete." *Id*. at 15:50-62 ("the string delete in an input stream such as input stream 36").

82. Martinez-Guerra teaches that the parser builds an internal "parse state data structure" that identifies the "valid next states" according to phrase structure rules encoded in the grammar. *Id.* at 18:31-48. As explained above, Martinez-Guerra's phrase structure rules together constitute a command parse tree because they are collectively a hierarchical data representation of the valid components of statements in a high-level language, and every component of each phrase structure rule has an action that will occur if that component is the last valid component parsed. The "phrase structure rules" within the grammar specify "the legal orderings of tokens in a language" (*id.* at 14:10-11) and are discussed in the specification of Martinez-Guerra in the context of textual rule specifications, such as "delete pathname-expert." *Id.* at 15:57, 14:32-36.

83. As input words, such as those of the generic command "delete /usr/extract/testing," are received from the user by the parser-translator, they are validated according to the phrase structure rules, which for rules like RULE1 together constitute a command parse tree. As Martinez-Guerra explains, the parser-translator keeps track of next legal choices in the phrase structure rules, and the token recognizer subcomponent of the parser-translator evaluates incoming words

received from the user against these next legal choices. *Id.* at 18:66-19:15; 19:25-35, 10:42-49. If the input word matches a next legal choice, then it is validated because it is a permissible choice. *Id.* at 18:66-19:11, 10:42-49.

84. As Martinez-Guerra illustrates, if the input word "delete" is received by the parser-translator, it will match the initial (root level) legal choice for the specification of RULE1 ("delete" in the rule specification "delete pathname-expert") because "delete" is a valid initial command term. *Id.* at 15:50-62. Martinez-Guerra further explains that "parser 32 begins with a first parse state and, consistent with phrase structure rules encoded in grammar 37, collects those tokens that are potentially valid next states corresponding to the first parse state." *Id.* at 19:25-29. Once validated, the token is accepted by the parser, and the parse state is updated to provide the token recognizer with a new list of next legal choices (*e.g.*, tokens that may permissibly follow the token "delete" in the command parse tree representation of RULE1). *Id.* at 19:5-7, 19:16-25, 19:31-35. If the user provides a valid pathname as the second input word, then the parser-translator would validate that token as a next legal choice in the command parse tree representation of the rule specification of RULE1 (*i.e.*, "pathname-expert" in the rule specification "delete pathname-expert"). *Id.* at 15:59-16:5, 20:40-49.

85. The parser-translator is thus configured to validate a generic command received from the user using a command parse tree.

3. [10B.2]: “the command parse tree configured for specifying valid generic commands relative to a prescribed generic command format and having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value,”

86. As discussed above with respect to claim elements 1C.1 and 1C.2, the grammar disclosed by Martinez-Guerra includes “a set of phrase structure rules and associated translation rules that define the syntactic order of a source and target language . . . .” *Id.* at 7:57-59, 16:15-16, 17:51-52. As discussed above, a person of ordinary skill in the art would understand that for any finite command language, Martinez-Guerra’s phrase structure rules constitute a command parse tree because they are collectively a hierarchical data representation of the valid components of statements in a high-level command language, and because every component of each phrase structure rule has an action that will occur if that component is the last valid component parsed. Phrase structure rules like “RULE1” (*id.* at 15:50-62) define a legal ordering of tokens according to a particular language—such as a high-level language containing generic commands (*id.* at 14:11-12)—and can be represented as a textual rule specification (“RULE1 → delete pathname-expert”). In the textual rule specification for RULE1, each token to the right of the arrow in a phrase structure rule is a component needed to form a complete statement; each of these is what the ’526 patent calls a “generic command component.”

87. Each component of the phrase structure rule has an appropriate corresponding action. If an input is invalid or a command is incomplete—for example, if only the word "delete" is received without any pathname to define the target of the operation, or if the subsequent pathname is wrongly written—that action may result in returning an error. *Id.* at 19:3-4. On the other hand, a valid input stream that matches and completes a phrase structure rule—*e.g.*, "delete /usr/extract/testing" (*id.* at 16:5)—will correspond to an action of the translation associated with the phrase structure rule. *Id.* at 8:60-65. The instruction to return an error that results from inputting either an invalid word or an incomplete sequence (*e.g.*, inputting "delete" without a target pathname), or the translation function that results from inputting a complete and valid sequence of tokens (*e.g.*, "delete /usr/extract/testing"), is what the '526 patent refers to as a "command action value." Such a command action value exists for each generic command component of a given phrase structure rule because, as input words are parsed, they may be invalid, resulting in an error, or valid, leading ultimately to the satisfaction of a phrase structure rule, thus resulting in a translation according to the corresponding translation function.

88. Each legal choice according to a phrase structure rule is thus an "element," as described in the '526 patent, having both a generic command component and a corresponding command action value.

4. [10B.3]: "the parser identifying one of the elements as a best match relative to the generic command; and"

89. As discussed above with respect to claim element 1C.3, , the token recognizer subcomponent of the parser-translator matches successive input words received from the user with the potential next legal choices in the parse state. *Id.* at 18:38-42, 18:66-19:3, 10:42-49. When the input word matches a next legal choice that completes a phrase—*e.g.*, a valid pathname matching the next legal choice "pathname-expert" in the rule specification RULE1 (*id.* at 15:50-62)—the parser identifies that next legal choice element as a best match for the generic command. *Id.* at 20:45-48.

5. [10C.1] "a plurality of translators configured for issuing commands for the management programs according to respective command formats,"

90. The parser-translator of Martinez-Guerra is configured for issuing commands for a management program according to respective command formats. Martinez-Guerra discloses that multiple implementations of the parser-translator are possible, including one in which "multiple instances of a parser-translator component, each with a corresponding grammar, provide multiple software applications with a common interface to high-level user language statements." *Id.* at 3:43-46. An implementation with multiple parser-translator components includes a "plurality of translators." Likewise, Martinez-Guerra describes an embodiment "wherein operation of [a] single parser-translator component is suita-

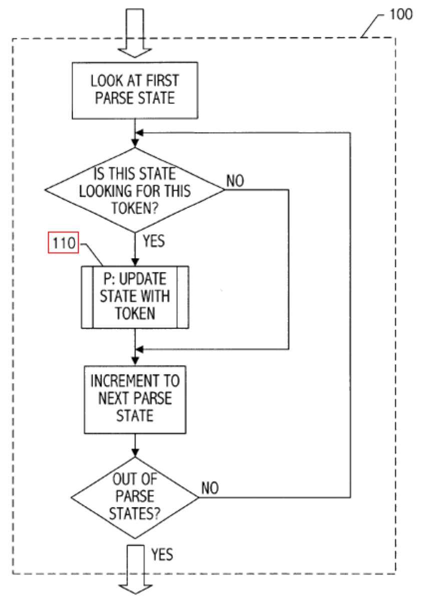
bly defined for each software application using a corresponding grammar encoding," *id.* at 3:47-53, which a POSA would understand is functionally equivalent to a plurality of translators.

91. As Martinez-Guerra explains, the disclosed translators are configured for issuing commands for management programs according to respective command formats. This is why each of the instances of parser-translators have "a corresponding grammar." *Id.* For example, Martinez-Guerra discloses that a translator subcomponent of the parser-translator issues an appropriate translation of the generic command "delete /usr/extract/testing" for the UNIX environment as "rm /usr/extract/testing." *Id.* at 15:59-16:5. This is a respective command format of the UNIX environment because "rm" is a UNIX shell command to delete the target file.

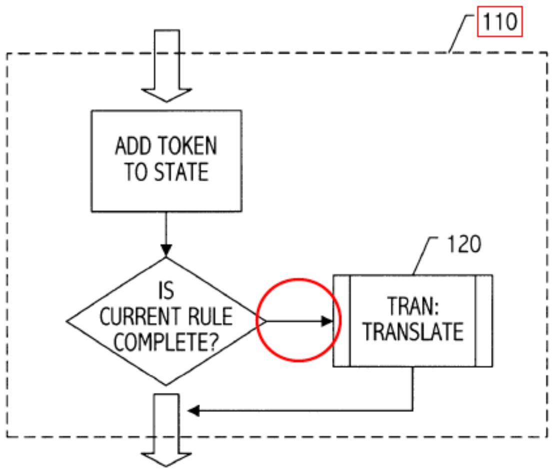
6. [10C.2] "the parser outputting a prescribed command to a selected one of the translators based on the identified one element."

92. Martinez-Guerra discloses that once a phrase structure rule is complete, which occurs when the final element of that rule is matched with a token from the input stream (*id.* at 20:45-48), the parser outputs a prescribed command to a translator based on that element. Figures 10 and 11 of Martinez-Guerra illustrate this process. Figure 10 discloses the process for determining whether a token matches the next legal choice of a phrase structure rule. *Id.* at 20:40-42. If the in-

put token is found in the command parse tree containing that phrase structure rule, then the parser updates the parse state by adding the token using process 110 of Figure 10. *Id.* at 20:49-58. Process 110 is illustrated in Figure 11, wherein the parser adds the token to the parse state and determines whether that addition has completed a phrase structure rule (*e.g.*, in the example discussed above, the final token, "pathname-expert," has been received). *Id.* at 20:42-45. If the token completes the rule, Figure 11 illustrates that the parser outputs a prescribed command to a translator, identified by the arrow directed to box 120 ("TRAN:"). *Id.* at 20:40-49. Box 120, illustrated in Figure 12, is the process undertaken by a translator subcomponent of the parser-translator. *Id.* at 20:45-48, Fig. 12.


100
LOOK AT FIRST PARSE STATE
IS THIS STATE LOOKING FOR THIS TOKEN?
NO
YES
110
P: UPDATE STATE WITH TOKEN
INCREMENT TO NEXT PARSE STATE
OUT OF PARSE STATES?
NO
YES

(Fig. 10 (annotation added).)



(Fig. 11 (annotation added).)

93. Because the command represented in Figure 11 is output by the parser as a result of matching the final element of a phrase structure rule, it is output

"based on the identified one element." The command is output to the appropriate translator to perform the translation. Martinez-Guerra discloses a "translation function identifier" specified for the element that completes a phrase structure rule, such as the "pathname-expert" element of "RULE1." *Id.* at 15:50-62, 16:11-16. The "translation function identifier" specifies where a translation function corresponding to a matched phrase structure rule can be located. *Id.* at 16:11-16, 20:61-64.

K. Claim 11

1. [11A.1] "The system of claim 10, wherein the parser further comprises a command word translation table configured for storing for each prescribed command word a corresponding token for identification of a matching token,"

94. As described above with respect to claim element 2B, Martinez-Guerra discloses that the "parser-translator 10 receives an input stream 6 from which token recognizer 11 extracts tokens in accordance with dictionary entries and rules encoded in the grammar 17 and in accordance with a current parse state." *Id.* at 9:35-40. The "dictionary entries" used by the parser-translator "describe all the tokens that a parser-translator . . . should recognize" (*id.* at 14:16-18), and as explained in relation to claim 2B above, parallel the '526 patent's description of the command word translation table "includ[ing] all the command words 26 that are valid according to the generic syntax." Exhibit 1001 at 3:45-46. One of ordinary skill in the art would understand from these disclosures and in view of

Martinez-Guerra's example of the "delete /usr/extract/testing" generic command, that the dictionary entries are a command word translation table coextensive with the operation of the parser-translator. Martinez-Guerra expressly notes that one of ordinary skill in the art would understand that subcomponent boundaries and their respective functionality in the disclosed parser-translator are subject to compositions other than those described in exemplary embodiments (Exhibit 1002 at 9:26-31, 22:37-42), and as such, would appreciate that the dictionary entries may be part of the parser-translator.

95. As explained with reference to claim 2B above, when the user submits the "delete /usr/extract/testing" command, the first input word "delete" is matched with a "regular token" entry in the dictionary having the same content ("delete"). As Martinez-Guerra explains, a "regular token is a fixed sequence of characters (for example, the string 'delete')." *Id.* at 15:2-4. The next (and final) input word the user submits is the pathname for the file to be deleted ("/usr/extract/testing" in the example). *Id.* at 15:57-16:5. This input word is what Martinez-Guerra refers to as an "expert token," and the parser-translator uses the dictionary entries to match the input word with the token "pathname-expert." *Id.* at 15:4-15, 15:63-16:5. The parser-translator then uses the matching token "pathname-expert" *instead of* the original input word "/usr/extract/testing" when searching for a match in the command parse tree.

2. [11A.2] "the parser configured for determining a presence of the matching token within the command parse tree for each input command word."

96. As described above with respect to claim element 2C, Martinez-Guerra discloses that the parser-translator determines a presence of the matching token within the command parse tree for each input word. In the example discussed in the previous limitation, after the parser-translator has determined the presence of the token corresponding to the input word "delete" in the command parse tree, the parser-translator then seeks to determine the presence of the second (and final) token. Martinez-Guerra explains that this is the token "pathname-expert," corresponding to a valid file path name supplied by the user, as explained in the previous limitation. *Id.* at 15:50-16:5, 20:40-45. The "pathname-expert" token would also be found in the command parse tree because it corresponds to the final token of the phrase structure rule RULE1. *Id.* at 15:57. The parser-translator of Martinez-Guerra thus determines the presence of a matching token in the command parse tree for each input command word.

L. Claim 12

1. "The system of claim 11, wherein the parser recursively traverses the command parse tree based on an order of the input command words for identification of the matching token within the identified one element."

97. Claim 12 recites the same method as claim 3, but specifies that the method is performed by the parser of independent claim 10. As discussed with

reference to claim 3, Martinez-Guerra teaches that claimed method is performed by the disclosed parser-translator. *Id.* at 20:31-34, 20:38-45. Because the method performed by the parser of claim 12 is identical to that recited in claim 3, this claim is disclosed by Martinez-Guerra as discussed above.

M. Claim 13

1. "The system of claim 12, wherein the parser validates at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command."

98. As discussed above with reference to claim 10, the parser-translator disclosed in Martinez-Guerra validates the generic command "delete /usr/extract/testing" in the example at column 15:50-62 because the token recognizer portion of the parser-translator determines, for each of the input words "delete" and "/usr/extract/testing" (where "/usr/extract/testing" is matched with the token "pathname-expert" in a dictionary entry) that the token corresponding to each input word is a "next legal choice" according to the phrase structure rule RULE1. By matching the final token ("pathname-expert") to the corresponding element in the command parse tree, which contains RULE1 ("delete pathname-expert"), Martinez-Guerra discloses that the parser-translator validates at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command. *Id.* at 20:31-34. Martinez-Guerra satisfies the limitation because validating the entire generic command "de-

lete /usr/extract/testing" necessarily validates "at least a portion of" the generic command. Further, Martinez-Guerra discloses that if an input word is invalid or a command is incomplete—for example, if only the word "delete" is received without any pathname to define the target of the operation—that action may result in returning an error. *See id.* at 19:3-4. In this circumstance, the parser-translator identifies a "best match" for less than the entire command.

N. Claim 14

1. [14A] "A computer readable medium having stored thereon sequences of instructions for executing a plurality of management programs according to respective command formats, the sequences of instructions including instructions for performing the steps of:"

99. Claim 14 recites the same method steps as claim 1, but is drafted in the form of a computer-readable medium for performing the claimed steps. Martinez-Guerra discloses that the disclosed invention may be carried out by instructions embodied in a computer-readable medium. *Id.* at 4:52-56, 5:51-53, 13:55-14:7, 22:45-57, 24:14-34 (claims 16-17), 25:61-26:10 (claims 32-33). Because the limitations of claim 14 following the preamble are identical to those of claim 1, claim 14 is anticipated for the same reasons that I have explained as to claim 1. My analysis of claim 14, limitations B-D, is the same as that for claim elements 1B-D, discussed above.

O. Claim 15

1. [15A] "The medium of claim 14, wherein the generic command includes at least one input command word, the validating step including:"

100. Claim 15 recites the same method steps as claim 2, discussed above, but is drafted as dependent from claim 14, a computer-readable medium for performing the claimed steps. Martinez-Guerra discloses that the invention may be carried out by instructions embodied in a computer-readable medium. *Id.* at 4:52-56, 5:51-53, 13:55-14:7, 22:45-57, 24:14-34 (claims 16-17), 25:61-26:10 (claims 32-33). Because the limitations of claim 15 following the preamble are identical to those of claim 2, claim 15 is anticipated for the same reasons that I have explained as to claim 2. My analysis of claim 15, limitations B-C, is the same as that for claim elements 2B-C, discussed above.

P. Claim 16

1. "The medium of claim 15, wherein the determining step includes recursively traversing the command parse tree based on an order of the input command words for identification of the matching token within the identified one element."

101. Claim 16 recites the same method as claim 3, discussed above, but is drafted as dependent from claim 14, a computer-readable medium for performing the claimed steps. Martinez-Guerra discloses that the invention may be carried out by instructions embodied in a computer-readable medium. *Id.* at 4:52-56, 5:51-53, 13:55-14:7, 22:45-57, 24:14-34 (claims 16-17), 25:61-26:10 (claims 32-33). Be-

cause claim 16 is otherwise identical to claim 3, claim 16 is anticipated for the same reasons that I have explained as to claim 3, discussed above.

Q. Claim 17

1. "The medium of claim 16, wherein the issuing step includes issuing the prescribed command based on a corresponding command key specified for the matching token within the identified one element."

102. Claim 17 recites the same method as claim 4, discussed above, but is drafted as dependent from claim 14, a computer-readable medium for performing the claimed steps. Martinez-Guerra discloses that the invention may be carried out by instructions embodied in a computer-readable medium. *Id.* at 4:52-56, 5:51-53, 13:55-14:7, 22:45-57, 24:14-34 (claims 16-17), 25:61-26:10 (claims 32-33). Because claim 17 is otherwise identical to claim 4, claim 17 is anticipated for the same reasons that I have explained as to claim 4, discussed above.

R. Claim 18

1. "The medium of claim 17, wherein the issuing step further includes accessing a prescribed translator configured for converting the generic command according to the corresponding command format into the prescribed command based on the corresponding command key."

103. Claim 18 recites the same method as claim 5, discussed above, but is drafted as dependent from claim 14, a computer-readable medium for performing the claimed steps. Martinez-Guerra discloses that the invention may be carried out by instructions embodied in a computer-readable medium. *Id.* at 4:52-56, 5:51-53,

13:55-14:7, 22:45-57, 24:14-34 (claims 16-17), 25:61-26:10 (claims 32-33). Because claim 18 is otherwise identical to claim 5, claim 18 is anticipated for the same reasons that I have explained as to claim 5, discussed above.

S. Claim 19

1. "The medium of claim 18, wherein the validating step including validating at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command, the issuing step including issuing the prescribed command based on the identified one element corresponding to the portion of the generic command."

104. Claim 19 recites the same method as claim 6, discussed above, but is drafted as dependent from claim 14, a computer-readable medium for performing the claimed steps. Martinez-Guerra discloses that the invention may be carried out by instructions embodied in a computer-readable medium. *Id.* at 4:52-56, 5:51-53, 13:55-14:7, 22:45-57, 24:14-34 (claims 16-17), 25:61-26:10 (claims 32-33). Because claim 19 is otherwise identical to claim 6, claim 19 is anticipated for the same reasons that I have explained as to claim 6, discussed above.

T. Claim 20

1. "The medium of claim 19, further comprising instructions for performing the step of executing the prescribed command within the corresponding selected one management program."

105. Claim 20 recites the same method as claim 7, discussed above, but is drafted as dependent from claim 14, a computer-readable medium for performing

the claimed steps. Martinez-Guerra discloses that the invention may be carried out by instructions embodied in a computer-readable medium. *Id.* at 4:52-56, 5:51-53, 13:55-14:7, 22:45-57, 24:14-34 (claims 16-17), 25:61-26:10 (claims 32-33). Because claim 20 is otherwise identical to claim 7, claim 20 is rendered obvious for the same reasons that I have explained as to claim 7, discussed above.

U. Claim 21

1. "The medium of claim 14, wherein the validating step including validating at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command, the issuing step including issuing the prescribed command based on the identified one element corresponding to the portion of the generic command."

106. Claim 21 recites the same method as claim 8, discussed above, but is drafted as dependent from claim 14, a computer-readable medium for performing the claimed steps. Martinez-Guerra discloses that the invention may be carried out by instructions embodied in a computer-readable medium. *Id.* at 4:52-56, 5:51-53, 13:55-14:7, 22:45-57, 24:14-34 (claims 16-17), 25:61-26:10 (claims 32-33). Because claim 21 is otherwise identical to claim 8, claim 21 is anticipated for the same reasons that I have explained as to claim 8, discussed above.

V. Claim 22

1. "The medium of claim 21, further comprising instructions for performing the step of executing the prescribed command within the corresponding selected one management program."

107. Claim 22 recites the same method as claim 9, discussed above, but is drafted as dependent from claim 14, a computer-readable medium for performing the claimed steps. Martinez-Guerra discloses that the invention may be carried out by instructions embodied in a computer-readable medium. *Id.* at 4:52-56, 5:51-53, 13:55-14:7, 22:45-57, 24:14-34 (claims 16-17), 25:61-26:10 (claims 32-33). Because claim 22 is otherwise identical to claim 9, claim 22 is rendered obvious for the same reasons that I have explained as to claim 9, discussed above.

W. Claim 23

1. [23A] "A system configured for executing a plurality of management programs according to respective command formats, the system comprising:

108. The preamble of claim 23 recites the same system as claim 10, discussed above. Accordingly, as previously demonstrated, Martinez-Guerra discloses the claimed system, wherein a parser-translator uses a high-level user language to interface with a plurality of software applications. *Id.* at 3:48-52. It does so by translating statements in a high-level language to "logically and syntactically correct directives for performing the desired data transformations or operations" using one or more software tools (*e.g.*, prescribed commands). *Id.* at 3:29-37. The dis-

closed method is carried out by software executed on a computer system. *Id.* at 9:10-13.

2. [23B.1] "means for validating a generic command received from a user,"

109. Martinez-Guerra discloses a parser-translator (*e.g.*, parser-translator 10 of Figure 1 and parser-translator 30 of Figure 3) that validates a generic command received from a user. As explained above with reference to claim 10B.1, Martinez-Guerra discloses that the parser-translator validates tokens—including tokens for a generic command such as "delete /usr/extract/testing"—received from a user via an input stream. *Id.* at Figs. 1, 3; 4:4-9; 9:15-23. The parser-translator of Martinez-Guerra thus includes the claimed "means for validating."

3. [23B.2] "the validating means configured for specifying valid generic commands relative to a prescribed generic command format and having elements each specifying at least one corresponding generic command component and a corresponding at least one command action value,"

110. As discussed above with respect to claim element 1C.1, Martinez-Guerra discloses that the parser-translator uses a grammar defining "a set of phrase structure rules and associated translation rules that define the syntactic order of a source and target language," (*id.* at 7:57-59, 16:15-16, 17:51-52) and dictionary entries "that describe all the tokens that a parser-translator (*e.g.*, token recognizer 31 of parser-translator component 30) should recognize" (*id.* at 14:16-18). As explained above with reference to claim 10B.2, the parser-translator is thus config-

ured for specifying valid generic commands relative to a prescribed generic command format.

111. As further discussed above with reference to claim element 10B.2, the command parse tree representation of the phrase structure rules created and maintained by the parser-translator component has elements specifying generic command components (*e.g.*, "delete," "/usr/extract/testing") and a corresponding command action value (*e.g.*, returning an error or performing a translation according to the completed phrase structure rule's corresponding translation function) for each token of a phrase structure rule. *See id.* at 8:60-65, 14:11-12, 14:19-21, 15:50-62, 17:47-58, 18:31-35, 19:3-4. For these reasons and those specified above as to claim element 10B.2, Martinez-Guerra discloses this limitation.

4. [23B.3] "the validating means identifying one of the elements as a best match relative to the generic command; and"

112. As explained above with reference to claim element 10B.3, Martinez-Guerra discloses that the parser-translator identifies the element corresponding to the final token needed to complete a phrase structure rule (*e.g.*, "pathname-expert") as the best match relative to the generic command (*e.g.*, "delete /usr/extract/testing"). Because "pathname-expert" is the last token needed to satisfy the phrase structure rule "RULE1," it is the best match relative to the generic

command "delete /usr/extract/testing." For these reasons and those specified above as to claim element 10B.3, Martinez-Guerra discloses this limitation.

5. [23C.1] "a plurality of translators configured for issuing commands for the management programs according to respective command formats,"

113. This limitation is identical to that of claim element 10C.1, thus, as previously demonstrated, Martinez-Guerra discloses this limitation.

6. [23C.2] "the validating means outputting a prescribed command to a selected one of the translators based on the identified one element."

114. This limitation is identical to that of claim element 10C.2, discussed above, except that claim element 23C.2 requires the claimed outputting to be performed by "the validating means" rather than "the parser," as in claim 10. As discussed above, the "validating means" includes the parser-translator and, accordingly, just as with the limitation in claim element 10C.2, Martinez-Guerra discloses the claimed validating means.

X. Claim 24

1. "The system of claim 23, wherein the validating means comprises a command word translation table configured for storing for each prescribed command word a corresponding token for identification of a matching token, the validating means configured for determining a presence of the matching token for each input command word."

115. Claim 24 recites the same system as claim 11, discussed above, but requires the limitation be performed by "validating means" rather than the "par-

ser," as in claim 11. As discussed in independent claim 23, the claimed "validating means" includes the parser-translator discussed in claim 10 and, accordingly, the analysis for this claim is the same as for claim 11.

Y. Claim 25

1. "The system of claim 24, wherein the validating means recursively validates each input command word based on an order of the input command words for identification of the matching token within the identified one element."

116. Claim 25 recites the same system as claim 12, previously described, but requires the limitation be performed by "validating means" rather than the "parser," as in claim 12. As discussed in independent claim 23, the claimed "validating means" includes the parser-translator discussed in claim 10 and, accordingly, the analysis for this claim is the same as for claim 12.

Z. Claim 26

1. "The system of claim 25, wherein the validating means validates at least a portion of the generic command by identifying the one element having the best match relative to the portion of the generic command."

117. Claim 26 recites the same system as claim 13, but requires the limitation be performed by "validating means" rather than the "parser," as in claim 13. As discussed in independent claim 23, the claimed "validating means" includes the parser-translator discussed in claim 10 and, accordingly, the analysis for this claim is the same as for claim 13.

118. In signing this declaration, I recognize that the declaration will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I also recognize that I may be subject to cross-examination in the case and that cross-examination will take place within the United States. If cross-examination is required of me, I will appear for cross examination within the United States during the time allotted for cross-examination.

119. I hereby declare under penalty of perjury under the laws of the United States of America that the foregoing is true and correct, and that all statements made of my own knowledge are true and that all statements made on information and belief are believed to be true. I understand that willful false statements and the like are punishable by fine or imprisonment or both under 18 U.S.C. § 1001.

Executed on November 4, 2015

[illegible]

Douglas W. Clark

Douglas W. Clark

Department of Computer Science
35 Olden Street
Princeton University
Princeton, NJ 08544-2087

2215 St. James Place
Philadelphia, PA 19103
(215) 563-5851

(609) 258-6314
doug@princeton.edu
http://www.cs.princeton.edu/~doug/

Education

Ph.D. in Computer Science, Carnegie-Mellon University, 1976
Xerox Graduate Fellow, 1973–1976

B.S. *magna cum laude* in Engineering and Applied Science, Yale University, 1972
Yale National Scholar, 1968–1972

Employment

Professor of Computer Science
Princeton University, since 1993

Visiting Professor of Computer and Information Science
University of Pennsylvania, spring 2003

Senior Consulting Engineer
Alpha Advanced Development, Digital Equipment Corp., 1991–1993

Visiting Lecturer
Division of Applied Sciences, Harvard University, 1990–1991

Consulting Engineer, then Senior Consulting Engineer
Advanced VAX Systems Engineering, Digital Equipment Corp., 1982–1990

Principal Engineer
Systems Architecture Group, Digital Equipment Corp., 1980–1982

Member of the Research Staff
Computer Science Laboratory, Xerox Palo Alto Research Center, 1976–1980

Professional activities

International Conference on Computer Design: program committee member, 2003

SIGMETRICS Conference on Measurement and Modeling of Computer Systems: program committee member, 1998, 1991, 1990

NSF Workshop on Critical Issues in Computer Architecture Research: workshop organizer, May 1996

Conference on Architectural Support for Programming Languages and Operating Systems: program chair, 1994; program committee member, 1992, 1988, 1982

ACM Transactions on Computer Systems: associate editor, 1983–1993

NSF Synthesis Engineering Education Coalition: national advisory committee member, 1990–1993

NSF Division of Microelectronic Information Processing Systems: advisory committee member, 1989–1992

International Symposium on Computer Architecture: program committee member, 1990, 1985; program committee co-chair, 1986

Publications

Wu, Q. Reddi, V., Wu, Y. Lee, J., Connors, D., Brooks, D., Martonosi, M., and Clark, D.W. "Dynamic Compiler Driven Control for Microprocessor Energy and Performance," *IEEE Micro* Special Issue: Top Picks from Computer Architecture Conferences, Vol. 26, No. 1, Jan./Feb., 2006.

Wu, Q. Reddi, V., Wu, Y. Lee, J., Connors, D., Brooks, D., Martonosi, M., and Clark, D.W. "A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance," *Proc. 38th IEEE/ACM International Symposium on Microarchitecture* (MICRO-38), Barcelona, Spain, November 2005. Best Paper Award.

Wu, Q., Juang, P., Peh, L.-S., Martonosi, M. and Clark, D.W. "Formal Control Techniques for Power-Performance Management," *IEEE Micro*, Vol 25, No. 5, Sept./Oct. 2005.

Juang, P., Wu, Q., Peh, L.-S., Martonosi, M. and Clark, D.W. "Coordinated, Distributed, Formal Energy Management of Chip Multiprocessors," *Proc. International Symposium on Low Power Electronics and Design* (ISLPED-05), San Diego, Aug. 2005.

Wallace, G., Anshus, O.J., Bi, P., Chen, H., Chen, Y., Clark, D., Cook, P., Finkelstein, A., Funkhouser, T., Gupta, A., Hibbs, M., Li, K., Liu, Z., Samanta, R., Sukthankar, R., and Troyanskaya, O. "Tools and Applications for Large-Scale Display Walls," *IEEE Computer Graphics*, Vol. 25, No. 4, July/Aug. 2005.

Q. Wu, P. Juang, M. Martonosi, and D. W. Clark, "Voltage and Frequency Control with Adaptive Reaction Time in Multiple-Clock-Domain Processors," *Proc. 11th International Symposium on High-Performance Computer Architecture* (HPCA-11), San Francisco, Feb. 2005.

Wu, Q., Juang, P., Martonosi, M., and Clark, D.W. "Formal Online Methods for Voltage/Frequency Control in Multiple Clock Domain Microprocessors," *Proc. Eleventh Int. Conf. on Architectural Support for*

Programming Languages and Operating Systems (ASPLOS-XI), Boston, Oct. 2004.

Y. Zhou, Y., Wang, L., Clark, D.W., and K. Li. "Thread Scheduling for Out-of-Core Applications with a Memory Server," ch. 9, *Scalable Input/Output: Achieving System Balance*, Daniel A. Reed, ed. MIT Press, 2004, pp. 233–253.

Juang, P., Skadron, K., Martonosi, M., Hu, Z., Clark, D.W, Diodato, P., and Kaxiras, S. "Implementing Branch-Predictor Decay Using Quasi-Static Memory Cells," *ACM Transactions on Architecture and Code Optimization*, Vol. 1, No. 2, June 2004.

Wu, Q., Pyatakov, A., Spiridonov, A., Raman, E., Clark, D.W., and August, D. "Exposing Memory Access Regularities Using Object-Relative Memory Profiling," *Proc. Second Ann. IEEE/ACM Int. Symp. on Code Generation and Optimization (CGO 2004)*, San Jose, March 2004.

Juang, P., Diodato, P., Kaxiras, S., Skadron, K., Hu, Z., Martonosi, M., and Clark, D.W. "Implementing Decay Techniques using 4T Quasi-Static Memory Cells," *Computer Architecture Letters 1*, Sept. 2002.

Hu, Z., Juang, P., Skadron, K., Martonosi, M., and Clark, D.W. "Applying Decay Strategies to Branch Predictors for Leakage Energy Savings," *Proc. 2002 Int. Conf. on Computer Design (ICCD2002)*, Sept. 2002.

Hu, Z., Juang, P., Diodato, P., Kaxiras, S., Skadron, K., Martonosi, M., and Clark, D.W. "Managing Leakage for Transient Data: Decay and Quasi-Static 4T Memory Cells," *Proc. 2002 Int. Symp. on Low Power Electronics and Design (ISLPED 2002)*, Aug. 2002.

Chen, Y., Chen, H., Clark, D.W., Liu, Z., Wallace, G., and Li, K. "Software Environments for Cluster-Based Display Systems," *IEEE Int. Symp. on Cluster Computing and the Grid (CCGrid 2001)*, May 2001.

Chen, Y., Clark, D.W., Finkelstein, A., Housel, T., and Li, K. "Automatic Alignment Of High-Resolution Multi-Projector Displays Using An Un-Calibrated Camera," *IEEE Visualization 2000*, Salt Lake City, Oct. 2000.

Skadron, K., Martonosi, M., and Clark, D.W. "A Taxonomy of Branch Mispredictions, and Alloyed Prediction as a Robust Solution to Wrong-History Mispredictions," *Int. Conf. on Parallel Architectures and Compilation Techniques*, Philadelphia, Oct. 2000.

Li, K., Chen, H., Chen, Y., Clark, D.W., Cook, P., Damianakis, S., Essl, G., Finkelstein, A., Funkhouser, T., Housel, T., Klein, A., Liu, Z., Praun, E., Samanta, R., Shedd, B., Singh, J.P., Tzanetakis, G., and Zheng, J. "Building and Using a Scalable Display Wall System," *IEEE Computer Graphics and Applications 20*, 4 (July/Aug. 2000), pp. 29-37.

Skadron, K., Martonosi, M., and Clark, D.W. "Speculative Updates of Local and Global Branch History: A Quantitative Analysis," *The Journal of Instruction Level Parallelism*, vol. 2, January 2000 (http://www.jilp.org/vol2).

Skadron, K., Ahuja, P.S., Martonosi, M., and Clark, D.W. "Branch Prediction, Instruction-Window Size, and Cache Size: Performance Tradeoffs and Sampling Techniques," *IEEE Transactions on Computers 48*, 3 (Nov. 1999), pp. 1260-1281.

Martonosi, M., Karlin, S., Liao, C., and Clark, D.W. "Performance Monitoring Infrastructure in the Shrimp Multicomputers," *Int. Journal of Parallel and Distributed Systems and Networks 2*, 3 (1999), pp. 126-133.

Liao, C., Martonosi, M., and Clark, D.W. "Experience with an Adaptive Globally-Synchronizing Clock Algorithm," *Proc. 11th ACM Symp. on Parallel Algorithms and Architectures*, Saint-Malo, France, June 1999.

Liao, C., Martonosi, M., and Clark, D.W. "An Adaptive Globally-Synchronizing Clock Algorithm and its Implementation on a Myrinet-based PC Cluster," *Proc. SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, Atlanta, May 1999.

Zhou, Y., Wang, L., Clark, D.W., and Li, K. "Thread Scheduling for Out-of-Core Applications with Memory Server on Multicomputers," *Proc. 6th Workshop on I/O in Parallel and Distributed Systems*, Atlanta, May 1999.

Skadron, K., Ahuja, P.S., Martonosi, M., and Clark, D.W. "Improving Prediction for Procedure Returns with Return-Address-Stack Repair Mechanisms," *Proc. 31st Annual Int. Symposium on Microarchitecture*, Dallas, Dec. 1998.

Wei, B., Clark, D.W., Felten, E.W., Li, K., and Stoll, G. "Performance Issues of a Distributed Frame Buffer on a Multicomputer," *Proc. 1998 Eurographics/SIGGRAPH Workshop on Graphics Hardware*, Lisbon, Aug./Sept. 1998.

Liao, C., Martonosi, M., and Clark, D.W. "Performance Monitoring in a Myrinet-Connected Shrimp Cluster," *Proc. 2nd SIGMETRICS Symposium on Parallel and Distributed Tools*, Oregon, August 1998.

Ahuja, P.S., Skadron, K., Martonosi, M., and Clark, D.W. "Multipath Execution: Opportunities and Limits," *Proc. 12th International Conference on Supercomputing*, Melbourne, July 1998.

Liao, C., Jiang, D., Iftode, L., Martonosi, M., and Clark, D.W. "Monitoring Shared Virtual Memory Performance on a Myrinet-based PC Cluster," *Proc. 12th International Conference on Supercomputing*, Melbourne, July 1998.

Blumrich, M.A., Alpert, R.D., Chen, Y., Clark, D.W., Damianakis, S.N., Dubnicki, C., Felten, E.W., Iftode, L., Li, K., Martonosi, M., and Shillner, R.A. "Design Choices in the SHRIMP System: An Empirical Study," *Proc. 25th Intl. Symposium on Computer Architecture*, Barcelona, June 1998, pp. 330-341.

Leung, C., Brooks, D., Martonosi, M., and Clark, D.W. "Power-Aware Architecture Studies: Ongoing Work at Princeton," *Proc. Power-Driven Microprocessor Design Workshop*, Barcelona, June 1998.

Wei, B., Stoll, G., Felten, E.F., Clark, D.W., and Li, K. "RAIN: Supporting Parallel Graphics on Paragon Multicomputers," *Proc. 13th Ann. Conf., Intel Supercomputer Users Group*, Albuquerque, June 1997.

Skadron, K. and Clark, D.W. "Design Issues and Trade-offs for Write Buffers," *Proc. Third Intl. Symposium on High-Performance Computer Architecture*, San Antonio, Feb. 1997, pp. 144-155.

Martonosi, M., Clark, D.W., and Mesarina, M., "The SHRIMP Hardware Performance Monitor: Design and Applications," *Proc. SIGMETRICS Symposium on Parallel and Distributed Tools*, Philadelphia, May 1996.

Felten, E.W., Alpert, R.D., Bilas, A., Blumrich, M.A., Clark, D.W., Damianakis, S., Dubnicki, C., Iftode, L., and Li, K., "Early Experience with Message-Passing on the SHRIMP Multicomputer," *Proc. 23rd Intl. Symposium on Computer Architecture*, Philadelphia, May 1996, pp. 296-307.

Skadron, K. and Clark, D.W., "Measuring the Effects of Retirement and Load-Service Policies on Write Buffer Performance," *Proc. 1996 Workshop on Performance Analysis and its Impact on Design (PAID)*,

IBM Austin Research Lab, Austin, March 1996.

Ahuja, P.S., Clark, D.W., and Rogers, A., "The Performance Impact of Incomplete Bypassing in Processor Pipelines," *28th IEEE/ACM Annual Int. Symp. on Micro-Arch. (MICRO-28)*, Ann Arbor, Nov. 1995, pp. 36-45.

Wei, B., Stoll, G., Clark, D.W., Felten, E.W., Li, K., and Hanrahan, P., "Synchronization for a Multi-Port Frame Buffer on a Mesh-Connected Multicomputer," *Proc. Parallel Rendering Symposium*, Atlanta, Oct. 1995, pp. 81-88.

Stoll, G., Wei, B., Clark, D.W., Felten, E.W., Li, K., and Hanrahan, P., "Evaluating Multi-Port Frame Buffer Designs for a Mesh-Connected Multicomputer," *Proc. 22nd International Symposium on Computer Architecture*, Santa Margherita Ligure, Italy, June 1995, pp 96-105.

Clark, D.W. and Weng, L.-J., "Maximal and Near-Maximal Shift Register Sequences: Efficient Event Counters and Easy Discrete Logarithms," *IEEE Transactions on Computers 43*, 5 (May 1994), pp. 560-568.

Bhandarkar, D. and Clark, D.W., "Performance from Architecture: Comparing a RISC and a CISC with Similar Hardware Organization," *Proc. Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, Santa Clara, CA, April 1991, pp. 310-319.

Clark, D.W., "Large-Scale Hardware Simulation: Modeling and Verification Strategies" (invited paper), Chapter 9 of *CMU Computer Science: A 25th Anniversary Commemorative* (R.F. Rashid, ed.), ACM Press/Addison-Wesley, 1991, pp. 219-234.

Clark, D.W., "Bugs are Good: A Problem-Oriented Approach to the Management of Design Engineering," *Research-Technology Management 33*, 3 (May-June 1990), pp. 23-27.

Clark, D.W., Bannon, P.J., and Keller, J.B., "Measuring VAX 8800 Performance with a Histogram Hardware Monitor," *Proc. 15th International Symposium on Computer Architecture*, Honolulu, June 1988, pp. 176-185.

Clark, D.W., "Pipelining and Performance in the VAX 8800," *Proc. Second International Symposium on Architectural Support for Programming Languages and Operating Systems*, Palo Alto, Oct. 1987, pp. 173-177.

Clark, D.W. and Emer, J.S., "Performance of the VAX-11/780 Translation Buffer: Simulation and Measurement," *ACM Transactions on Computer Systems 3*, 1 (Feb. 1985), pp. 31-62.

Emer, J.S. and Clark, D.W., "A Characterization of Processor Performance in the VAX-11/780," *Proc. 11th International Symposium on Computer Architecture*, Ann Arbor, June 1984, pp. 301-310.

Clark, D.W., "Cache Performance in the VAX-11/780," *ACM Transactions on Computer Systems 1*, 1 (Feb. 1983), pp. 24-37.

Levy, H.M. and Clark, D.W., "On the Use of Benchmarks for Measuring System Performance," *ACM Computer Architecture News 10*, 6 (Dec. 1982), pp. 5-8.

Clark, D.W. and Levy, H.M., "Measurement and Analysis of Instruction Use in the VAX-11/780," *Proc. 9th International Symposium on Computer Architecture*, Austin, April 1982, pp. 9-17.

Clark, D.W., Lampson, B.W., and Pier, K.A., "The Memory System of a High-Performance Personal Computer," *IEEE Transactions on Computers C-30*, 10 (Oct. 1981), pp. 715-733.

Clark, D.W. and Strecker, W.D., “Comments on ‘The Case for the Reduced Instruction Set Computer’ by Patterson and Ditzel,” *ACM Computer Architecture News 8*, 6 (Oct. 1980), pp. 34-38.

Bobrow, D.G. and Clark, D.W., “Compact Encodings of List Structure,” *ACM Transactions on Programming Languages and Systems 1*, 2 (Oct. 1979), pp. 266-286.

Clark, D.W., “Measurements of Dynamic List Structure Use in Lisp,” *IEEE Transactions on Software Engineering SE-5*, 1 (Jan. 1979), pp. 51-59.

Clark, D.W. and Green, C.C., “A Note on Shared List Structure in Lisp,” *Information Processing Letters 7*, 6 (Oct. 1978), pp. 312-314.

Clark, D.W., “A Fast Algorithm for Copying List Structures,” *Communications of the ACM 21*, 5 (May 1978), pp. 351-357.

Clark, D.W. and Green, C.C., “An Empirical Study of List Structure in Lisp,” *Communications of the ACM 20*, 2 (Feb. 1977), pp. 78-87.

Clark, D.W., *List Structure: Measurements, Algorithms, and Encodings*, Ph.D. thesis, Dept. of Computer Science, Carnegie-Mellon University, Aug. 1976.

Clark, D.W., “An Efficient List-Moving Algorithm Using Constant Workspace,” *Communications of the ACM 19*, 6 (June 1976), pp. 352-354.

Clark, D.W., “A Fast Algorithm for Copying Binary Trees,” *Information Processing Letters 4*, 3 (Dec. 1975), pp. 62-63.

Clark, D.W., “Hardware Systems in the Core Curriculum of a Computer Science Ph.D. Program,” *Proc. ACM Fourth Technical Symposium on Computer Science Education*, Detroit, Feb. 1974, pp. 106-110.